数据通信网络技术

曹　林　张月霞　编著

電子工業出版社
Publishing House of Electronics Industry
北京・BEIJING

VISIT…

内 容 简 介

本书依托大学生科研训练与工程实践，详细讨论了数据通信网络技术相关理论及热点技术。全书共分10章，第1章概述了数据通信网络基础；第2章探讨了数据链路层与网路层协议；第3章研究了传输层协议；第4章总结了计算网络相关应用；第5章阐述了固定电话网、数字数据网、排队论等常规通信网技术；第6章分析了智能网技术；第7章讨论了防火墙、病毒、加密等网络安全技术；第8章研究了计算机网络编程技术；第9～10章，结合工程实际探讨了通信网OPNET仿真技术，并重点讨论了基于OPNET的排队模型及无线建模。

本书可供从事数据通信、计算机网络及相关研究和开发的教师、研究生及工程技术人员参考。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

数据通信网络技术 / 曹林，张月霞编著. —北京：电子工业出版社，2015.7
ISBN 978-7-121-26407-8

Ⅰ. ①数… Ⅱ. ①曹… ②张… Ⅲ. ①数据通信—通信网 Ⅳ. ①TN919.2

中国版本图书馆CIP数据核字（2015）第138300号

策划编辑：董亚峰
责任编辑：郝黎明
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编 100036
开　　本：787×1 092　1/16　印张：21.75　字数：550千字
版　　次：2015年7月第1版
印　　次：2015年7月第1次印刷
定　　价：49.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前 言

数据通信网络技术是21世纪发展最快、影响最深远的技术，其影响已深入到社会经济的各个领域。近年来，我国计算机技术发展日新月异，网络传输的效率大大提升，基于网络环境中的互联网设备朝着集成化及智能化的方向完善，数据通信网络技术从以下方面发展。

（1）移动、联通、电信公司将朝着 5G 方向发展，从而满足用户的信息交流及信息资源共享需求。

（2）宽带无线接入技术进一步完善，随着 WiFi 热点的逐渐变大，使我国宽带局域网的发展进一步加大，在数据通信与计算机网络技术充分融合的背景下，宽带无线接入技术将进一步得到完善。

（3）光通信将获得巨大发展前景，包括 ASON 能够获得充分有效的利用以及带宽资源的管理力度将加大，从而使光通信技术更具实用价值。

本书讨论了数据通信网络技术相关理论及热点技术，包括 TCP/IP 数据链路层、网络层、传输层、应用层等协议；防火墙、病毒、加密等网络安全技术；固定电话网、数字数据网、排队论等常规通信网技术，以及宽带智能网技术。此外本书结合工程实践，探讨了计算机网络编程技术及通信网 OPNET 仿真技术，并重点讨论了基于 OPNET 的排队模型及无线建模。本书共分 10 章，其中第 2、3、4、5、8 章由曹林撰写，第 1、6、7、9、10 由张月霞撰写，全书由曹林统稿。本书所涉及的主要内容来源于数据通信网络技术相关实践，同时

广泛参考了国内外同行的研究成果。本书的出版得到了北京市教委本科生培养-大学生科研训练（项目号：PXM2014-014224000079）、北京市属高校青年拔尖人才培育计划（项目号：CIT&TCD201304119、CIT&TCD201504058）、国家自然科学基金重点项目（项目号：51334003）、人才培养项目—学科与研究生教育水平提高项目（项目号：5111524100）等科研项目的资助，在此一并表示感谢。

由于时间仓促，书中难免存在不足，欢迎读者对本书批评指正。

目 录

第 1 章　数据通信基础 …… 1

1.1　数据通信网发展历史 …… 1

1.1.1　数据通信网的发展过程 …… 1

1.1.2　中国数据通信网的发展 …… 3

1.2　数据通信网的分类 …… 4

1.3　数据通信网的拓扑结构 …… 6

1.3.1　总线型拓扑结构 …… 6

1.3.2　星型拓扑结构 …… 6

1.3.3　环型拓扑结构 …… 7

1.3.4　树型拓扑结构 …… 7

1.3.5　网状型拓扑结构 …… 8

1.3.6　混合型拓扑结构 …… 8

1.3.7　蜂窝型拓扑结构 …… 9

1.4　网络体系结构 …… 10

1.4.1　网络协议与网络体系结构 …… 10

1.4.2　OSI 模型 …… 11

1.4.3　TCP/IP 模型 …… 14

小结 …… 16

参考文献 …… 16

第 2 章　数据链路层和网络层 ……17
2.1　数据链路层 ……17
2.1.1　数据链路和帧 ……18
2.1.2　3 个基本问题 ……19
2.1.3　数据链路控制 ……24
2.1.4　数据链路控制协议举例 ……30
2.2　网络层 ……36
2.2.1　网际协议 ……36
2.2.2　路由协议 ……53
2.2.3　虚拟专用网和网络地址转换 ……64
2.2.4　下一代网际协议 IPv6 ……65
小结 ……72
参考文献 ……72
第 3 章　传输层 ……73
3.1　传输层简介 ……73
3.1.1　传输层的端-端通信及与 QoS 的关系 ……73
3.1.2　传输层协议简介 ……73
3.1.3　传输服务与 QoS ……75
3.2　UDP ……77
3.2.1　UDP 报文的格式 ……77
3.2.2　UDP 的封装与协议的分层 ……78
3.2.3　UDP 的复用、分解与端口 ……79
3.2.4　UDP 的可靠性 ……79
3.3　TCP ……80
3.3.1　TCP 简介 ……80
3.3.2　TCP 的编号与确认 ……82
3.3.3　TCP 报文段 ……83
3.3.4　TCP 连接管理 ……87
3.3.5　TCP 状态机 ……90
3.3.6　TCP 重传机制 ……94
3.3.7　TCP 流量控制 ……97
3.3.8　TCP 拥塞控制简介 ……101
3.3.9　TCP 差错控制 ……104
3.3.10　TCP 的计时器 ……106
3.3.11　TCP 的安全 ……108
3.3.12　TCP 的主要特点 ……110

小结 …… 112
参考文献 …… 112
第 4 章　应用层 …… 113
4.1　应用层简介 …… 113
4.1.1　应用层协议及其与低层协议的关系 …… 113
4.1.2　网络应用进程交互的 C/S 模式 …… 114
4.1.3　进程通信中 C/S 模式的实现方法 …… 116
4.2　InternetDNS 简介 …… 118
4.2.1　DNS 原理 …… 118
4.2.2　域名的分级 …… 118
4.2.3　InternetDNS …… 119
4.2.4　域名和地址映射 …… 121
4.3　文件传输协议 …… 121
4.3.1　文件传输协议 …… 121
4.3.2　镜像系统 …… 123
4.4　远程登录协议 …… 124
4.5　电子邮件协议 …… 126
4.5.1　电子邮件简介 …… 126
4.5.2　电子邮箱和地址 …… 127
4.5.3　电子邮件的格式 …… 128
4.5.4　SMTP 简介 …… 129
4.6　万维网 …… 132
4.6.1　万维网的工作原理 …… 132
4.6.2　统一资源定位符（URL） …… 137
4.6.3　超文本传送协议（HTTP） …… 138
4.6.4　超文本标记语言（HTML） …… 141
4.6.5　动态 Web 文档技术 …… 144
4.6.6　活动 Web 文档技术 …… 148
小结 …… 149
第 5 章　网络安全 …… 151
5.1　防火墙 …… 151
5.1.1　防火墙简介 …… 151
5.1.2　防火墙技术 …… 155
5.2　信息加密 …… 163
5.2.1　信息加密简介 …… 163

5.2.2 数据加密标准 DES 与 IDEA 算法 …… 165
5.2.3 病毒防护公开密钥算法 …… 167
5.2.4 密钥管理和交换技术 …… 171
5.2.5 密码分析与攻击 …… 173
5.3 病毒防护 …… 175
5.3.1 病毒简介 …… 175
5.3.2 病毒原理 …… 178
小结 …… 185
第 6 章 通信网 …… 186
6.1 固定电话网 …… 186
6.1.1 固定电话网的起源 …… 186
6.1.2 固定电话网的组成 …… 187
6.1.3 固定电话网的特点 …… 188
6.1.4 固定电话网组网 …… 188
6.1.5 固定电话网编号 …… 191
6.2 X.25 分组网 …… 192
6.3 数字数据网 DDN …… 193
6.4 帧中继 FR 网 …… 196
6.5 VoIP 技术 …… 197
6.6 接入网 …… 200
6.6.1 接入网简介 …… 200
6.6.2 有线接入网 …… 203
6.6.3 无线接入技术 …… 207
6.7 排队论基础 …… 207
6.7.1 排队模型基本概念 …… 207
6.7.2 泊松过程 …… 208
6.7.3 M/M/1 排队模型 …… 210
小结 …… 212
参考文献 …… 212
第 7 章 智能网 …… 214
7.1 智能网概述 …… 214
7.1.1 什么是智能网 …… 214
7.1.2 智能网的基本特点 …… 215
7.1.3 智能网和其他业务网的关系 …… 216
7.1.4 智能网的概念模型 …… 216
7.2 智能网的结构 …… 219

7.2.1 业务交换点 …… 220
7.2.2 业务控制点 …… 220
7.2.3 智能外设 …… 222
7.2.4 业务管理点 …… 223
7.2.5 业务生成环境 …… 223
7.3 移动网与智能网的结合 …… 224
7.3.1 CAMEL 结构、协议及业务 …… 224
7.3.2 CAMEL 网络结构 …… 225
7.3.3 CAMEL 协议 …… 225
7.4 宽带智能网 …… 226
7.4.1 智能网与宽带综合业务数字网的互连 …… 226
7.4.2 宽带智能网的体系结构与呼叫模型 …… 227
7.5 智能网与因特网互连 …… 229
7.5.1 IN 与 Internet 的互连方案 …… 229
7.5.2 IN/Internet 互连的安全问题 …… 231
7.6 智能网与电信管理网 …… 231
7.6.1 TMN 的产生和发展 …… 232
7.6.2 TMN 的基本概念 …… 233
7.7 智能网的发展趋势 …… 235
7.7.1 现有智能网技术的缺陷 …… 235
7.7.2 下一代智能网的重要特征 …… 236
小结 …… 238
参考文献 …… 238
第 8 章 计算机网络编程基础 …… 239
8.1 线程同步与异步套接字编程 …… 239
8.1.1 事件对象 …… 239
8.1.2 关键代码段及其应用 …… 246
8.1.3 基于消息的异步套接字 …… 253
8.2 进程间通信 …… 266
8.2.1 匿名管道 …… 267
8.2.2 命名管道 …… 278
小结 …… 290
第 9 章 通信网编程——OPNET …… 291
9.1 仿真技术 …… 291
9.1.1 仿真的定义 …… 291

9.1.2 仿真的分类 …… 292
9.1.3 网络仿真 …… 292
9.1.4 仿真工具 …… 294
9.2 OPNET 概述 …… 295
9.2.1 OPNET 历史和现状 …… 295
9.2.2 OPNET 的系列产品 …… 296
9.2.3 OPNET Modeler 仿真平台 …… 299
9.3 OPNET Modeler 运行环境 …… 300
9.3.1 系统需求 …… 301
9.3.2 OPNET Modeler 安装 …… 301
9.3.3 OPNET Modeler 开发环境 …… 302
9.4 OPNET Modeler 编程基础 …… 304
9.4.1 Modeler 编程概述 …… 304
9.4.2 Modeler 文件操作 …… 306
9.4.3 实例：建立简单星型网络 …… 308
9.5 OPNET 程序调试 …… 315
9.5.1 查看 OPNET 日志文件 …… 316
9.5.2 使用 OPNET Debugger 调试 …… 318
9.5.3 OPNET 与 Visual C++联合调试 …… 319
9.5.4 ODB 常见错误及问题 …… 321
小结 …… 323
参考文献 …… 323
第 10 章 综合案例——OPENT 编程 …… 324
10.1 排队模型 …… 324
10.2 无线建模 …… 328
参考文献 …… 336

第 1 章 数据通信基础

1.1 数据通信网发展历史

数据通信网是通信与计算机技术相结合的产物，为计算机和其他电子设备之间提供了一种新的通信方式。为了实现不同设备之间的通信，通信双方必须遵守相同的协议，并通过某种介质以实现在计算机或者电子设备之间的数据传输。

1.1.1 数据通信网的发展过程

19 世纪中叶发明的电报与电话机标志着现代通信技术的开始；20 世纪 40 年代发明的计算机标志着现代信息处理技术的开始；20 世纪 60 年代后期发明的计算机网络则使人类进入了信息时代。而且随着计算机的广泛使用，由计算机和各种电子设备组成的数据通信网成为未来发展的必然趋势。数据通信网从诞生到现在大致可以分为以下 4 个阶段。

1．第一阶段——单机多终端系统

数据通信网最早出现在 20 世纪 50～60 年代，这种通信网利用硬件设备使一台中心计算机通过通信线路与许多终端相连，这样用户可以通过通信线路共享一台计算机，并进行数据通信。这种简单的网络结构形成了数据通信网的雏形，如图 1-1 所示。1952 年，美国半自动地面防空系统的科研人员首次把远程雷达和其他测量设备的信息，通过通信线路汇接到一台计算机上，进行集中处理和控制。但是这种系统的主机由于既要进行数据处理，又要承担通信工程，负荷较重，响应时延较长，导致系统效率较低。同时，每个远程终端都分散在各地，它们与主机之间通信都需要一条单独的通信链路，通信线路利用率较低，速率也比较低。

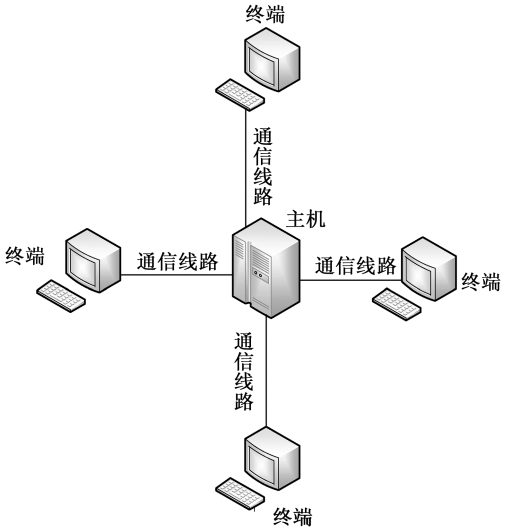


图 1-1　单机多终端系统

2．第二阶段——多机互连系统

单机多终端系统之间是相互独立的，不同的主机之间相互没有联系。随着设备价格的降低，计算机得到了更为广泛的应用，不同地域的远距离的多台计算机之间彼此间共享资源、交流信息的需求应运而生。20 世纪 60～70 年代就产生了多机互连系统，这是数据通信网络的雏形，此时的数据通信网分为资源子网和通信子网两个部分，如图 1-2 所示。多个单机多终端系统构成了资源子网，负责提供数据通信网的内容和数据处理；各个单机多终端系统之间通过通信设备和传输介质相互通信；由通信设备、传输介质和通信协议等构成通信子网，为资源子网提供信息传输服务。

这个阶段最有影响力的网络是1969年9月由美国国防部高级研究计划所和十几个计算机中心一起研制出的 ARPANET 网络。ARPANET 网络最初只包含加利福尼亚大学洛杉矶分校（UCLA）、加利福尼亚大学圣巴巴拉分校（UCSB）、犹他大学（Utah）和斯坦福研究所（SRI），这 4 个站点，后来一直发展到 15 个站点、23 台主机，实现了将多个大学、科研机构和公司的计算机相连，实现了资源共享。

3．第三阶段——统一协议的数据通信网

多机互连系统阶段，各个厂家都采用各自设计的体系结构，不同厂家的设备之间无法互连互通，即使同一厂家的不同阶段的设备之间也无法互连互通，这就严重影响了数据通信网发展的规模。而且伴随着通信技术和计算机技术的发展，各种不同网络之间互连的需求越来越迫切。为了实现各种不同网络之间的通信，急需研制具有国际统一标准协议的数据通信网络。20 世纪 70～80 年代，国际标准化组织（International Organization for Standardization，ISO）公布了开放系统互连（Open System Interconnection，OSI，详见 1.4 节）参考模型，规定了可以互连的计算

机系统之间的通信协议，定义了异种机联网的标准框架，为连接分散的“开放”系统、更大范围内的共享计算机资源进行多样化的数据通信奠定了基础。

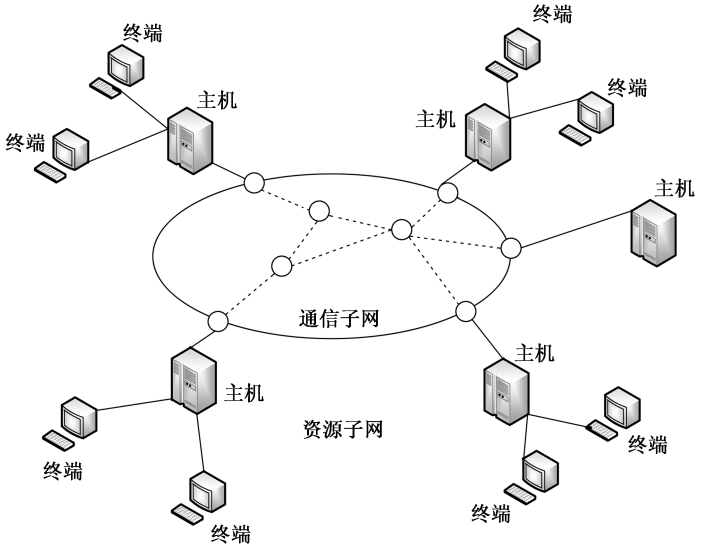


图 1-2　多机互连系统

1974 年，斯坦福大学的温顿·瑟夫和罗伯特·卡恩提出了 TCP/IP。1983 年，该协议在 ARPANET 网络上应用，使得 TCP/IP 成为网络标准，这标志着全球 Internet 的诞生。

4．第四阶段——高速数据通信网络

数据通信网高速发展的阶段是从 20 世纪 90 年代，光纤技术以其低成本、高速率的特点而被作为传输介质得到广泛应用。光纤的广泛应用使得广域网主干带宽从 10 Gb/s、40 Gb/s 达到 100 Gb/s，网络不但可以传输数据，也可以传输语音、视频等应用，并且通过智能化程序很高的网络管理软件和硬件实现网络的智能化，使网络进入一个崭新的信息高速公路时代。

1.1.2　中国数据通信网的发展

我国互联网的起步比国际上略晚，是从 20 世纪 70～80 年代起步的，但发展却极其迅速。1975 年，我国开始了远程终端连机实验，并在 1980 年开始建设计算机广域网。随即数据通信网在我国掀起了一阵热潮，开始飞速发展。其发展历史主要分为以下 3 个阶段。

- 第一阶段是 1986—1994 年，这个阶段是中国数据通信网的开始阶段，其标志性事件是 1986 年钱天白教授成功地给德国发送了中国的第一封电子邮件。钱教授也被称为中国的 Internet 之父。其后在 1989 年，中国的 ChinaPAC（X.25）公用数据网基本开通，该网络能够与法国、德国等的公用数据网络（X.25）实现国际连接（X.75）。1993 年，中国科学院高能物理

研究所开通了一条 64kb/s 的国际数据通道，实现了网络数据通信。

- 第二阶段是 1994—1995 年。虽然两年的时间很短，但是在这段时间中，我国教育科研网络发展飞跃到了一个新的高度。1989 年 8 月，中国科学院承担了原国家计划委员会立项的“中关村地区教育与科研示范网络”（NCFC）——中国科学技术网（CSTNET）前身的建设。1994 年 4 月，NCFC 率先与美国 NSFNET 直接互连，实现了中国与 Internet 全功能网络连接，标志着中国最早的国际互联网络的诞生。此后又建立了中国教育和科研计算机网（CERNET）。
- 第三阶段是从 1995 年至今，是互联网的应用阶段。1995 年 1 月，邮电部电信总局分别在北京、上海设立的通过美国 Sprint 公司接入美国的 64Kb/s 专线开通，并且通过电话网、DDN 专线及 X.25 网等方式开始向社会提供 Internet 接入服务。到 1996 年 9 月机械电子部的 ChinaGBN 开通，各地的 Internet 服务提供者的快速增长使得互联网真正走进了我国百姓的生活。时至今日，电子商务、即时通信、在线游戏、网络小说等互联网应用改变了我国人民的生活，也成为我国人民生活的一部分，更为国民经济的发展提供了巨大的助力。

迄今为止，我国建造的和 Internet 互连的全国范围的公用计算机网络有中国公用计算机互联网（ChinaNET）、CERIVET、CSTNET、中国联通互联网（UNINET）、中国国际经济贸易互联网（CIETNET）、中国移动互联网（CMNET）。

1.2 数据通信网的分类

数据通信网按照不同属性有很多分类方式。

1．按网络覆盖的地理范围进行分类

- 局域网（Local Area Network，LAN）：是指在某一区域内将各种计算机、外围设备和数据库等互相连接起来组成的计算机通信网。其主要特点是覆盖范围比较小，一般在几千米以内，适用于一个部门、一所学校或一栋办公楼；成本低；传输速率高；方便灵活，容易组网；传输时延较小。
- 城域网（Metropolitan Area Network，MAN）：是指在一个城市范围内建立的数据通信网，覆盖范围由几千米到几十千米。其主要以光纤作为主要介质，拓扑结构比较规则，传输速率比较高，传输效率较高，差错率较低。
- 广域网（Wide Area Network，WAN）：是指覆盖范围较大的网络，其覆盖范围远大于局域网和城域网，它能连接多个城市或国家，或横跨几个洲并能提供远距离通信，形成国际性的远程网络。广域网的通信子网可以利用公用分组交换网、卫星通信网和无线分组交换网，它将分布在不同地区的局域网或计算机系统互连起来，达到资源共享的目的。例如，互联网是世界范围内最大的广域网。广域网是由许多交换机组成的，交换机之间采用点到点线路连接，几乎所有的点到点通信方式都可以用来建立广域网，包括租用线路、光纤、微波、卫星信道。而广域

网交换机实际上就是一台计算机，由处理器和输入/输出设备进行数据包的收发处理。

2．按网络拓扑结构进行分类

按照网络拓扑结构的不同，可将数据通信网分为总线型网络、星型网络、环型网络、树型、网状型、混合型和蜂窝型等。

3．按通信传播方式进行分类

- 广播式网络：网络中所有终端均通过公共信道连接，任何一个终端可以通过公共信道将信息发送给网络中所有的终端。当终端收到分组信息时，首先检查分组中目的地址是否与自己的地址相符合，若符合，将接收该分组；若不符合，将丢弃该分组。局域网、无线网和总线型网络基本上都是广播式网络。
- 点对点网络：在这种网络中，如果源节点和目的节点之间有直连链路，信息分组将可以直接传送至对方。如果源节点和目的节点之间没有直接链路，则信息分组将通过中间节点转发至目的节点。在庞大复杂的 Internet 中，中间节点可以有很多种不同的选择，中间节点的选择由网络中的路由算法决定。

4．按经营网络的主管部分进行分类

- 公用网：是由国家主管部门经营管理的网络，如 ChinaNET、公共交换电话网（Public Switched Telephone Network，PSTN）等。
- 专用网：是根据各专业部门内部通信的需要而组成的内部通信网络，一般为某个单位或某一系统组建，该网一般不允许系统外的用户使用，如银行、公安、铁路等系统建立的网络是本系统专用的。

5．按数据通信网的传输介质进行分类

- 有线网络：指采用同轴电缆、双绞线、光纤等有线介质连接的网络。同轴电缆和双绞线的连接方式价格比较低，安装方便，但是传输速率较低，传输距离较短。光纤是利用光在玻璃或塑料制成的纤维中的全反射原理而制成的光传播介质，一般在发射端采用发光二极管将光脉冲传送至光纤，接收装置使用光敏元件检测脉冲。由于光纤通信损耗较少，在日常生活中的应用越来越多。
- 无线网络：指采用微波、红外线、无线电等电磁波作为传输介质的网络。其由于联网方式灵活而备受青睐，现在已经得到了广泛应用。

6．按交换方式进行分类

按照网络中信息交换的方式，可以把数据通信网分为电路交换网、分组交换网、报文交换网、帧中继交换网、信元交换网和 IP 交换网等。

1.3 数据通信网的拓扑结构

拓扑学是几何学的一个分支，它将物体抽象为与其大小无关的点，将连接物体的线路抽象为与距离无关的线，进而研究点、线、面之间的关系。数据通信网的拓扑结构用网络的节点与线的几何关系来表示网络结构。

数据通信网的拓扑结构有很多种，常见的有总线型、星型、环型、树型、网型、混合型和蜂窝型等。下面将对各种拓扑结构进行介绍与说明。

1.3.1 总线型拓扑结构

在总线型拓扑结构（图 1-3）中，所有的节点共享一条数据通道，总线上的任一节点发出的信息都可以被其他节点收到。因为多个节点连接到一条共享数据通道上，所以必须采取某种方法分配信道，以决定哪个节点可以发送数据。如果两个或两个以上的节点同时向总线发送信号，则信号会发生冲突。

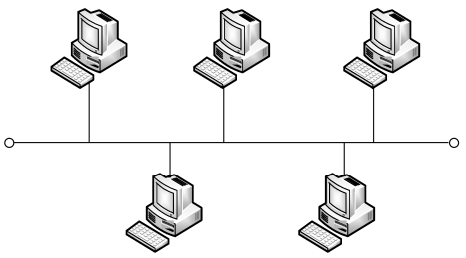


图 1-3 总线型拓扑结构图

总线型结构的优点是所需电缆数量较少；结构简单，无源工作有较高可靠性；易于扩充。总线型结构的缺点是传输距离有限，通信范围受到限制；故障诊断和隔离比较困难；总线上任一点故障会使整个网络不能正常工作；入网节点越多，总线负担越重，冲突概率越大，传输效率越低。总而言之，总线型拓扑结构适用于计算机数目相对较少的局域网络，典型的总线型局域网有传统以太网。

1.3.2 星型拓扑结构

星型拓扑结构（图 1-4）采用集中控制方式。在星型拓扑结构中，各节点通过线路与中心节点相连，中心节点对各设备间的通信和信息交换进行集中控制和管理。每一个要发送数据的节点都将要发送的数据发送至中心节点，再由中心节点负责将数据送到目的节点。因此，星型拓扑结构对中心节点要求较高，信息处理能力较强，也比较复杂，而其他节点只需要满足链路的简单通信要求。

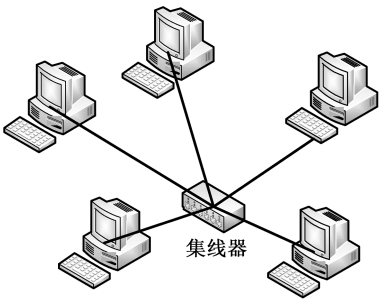


图 1-4 星型拓扑结构图

星型拓扑结构的优点如下。

1）控制简单，易于网络监控和管理。

2）故障诊断和隔离容易，中央节点对连接线路可以逐一隔离，并进行故障检测和定位，单个连接点的故障只影响一台设备，不会影响全网。

3）方便服务，中央节点可以方便地对各个站点提供服务和网络重新配置。

星型拓扑结构的缺点如下。

1）需要耗费大量的电缆，安装、维护的工作量也骤增。

2）中央节点负担重，一旦发生故障，则全网受影响。

3）各站点的分布处理能力较低。

星型拓扑结构是在生活中应用最广的网络拓扑结构，一般的学校、办公楼都采用这种网络拓扑组建单位的计算机网络。

1.3.3　环型拓扑结构

在环型拓扑结构（如图 1-5 所示）中，终端设备通过转发器接入网络，每个转发器仅与两个相邻的转发器有直接的物理线路。环形网的数据传输具有单向性，一个转发器发出的数据只能被另一个转发器接收并转发，所有的转发器及其物理线路构成了一个环状的网络系统，所以网络中的信息流是定向的，网络的传输延时也是确定的。环型拓扑结构的优点是路径选择与控制软件简单；传输延迟确定，易于实现。环型拓扑结构的缺点是扩展性差，增删节点操作困难；可靠性差，环路上任一点一旦出故障，会使整个网络瘫痪。因此可用双通道环来提高可靠性，如图 1-6 所示。

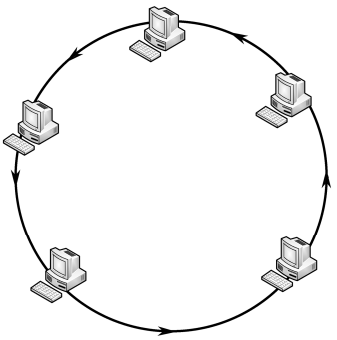


图 1-5　环形拓扑结构图

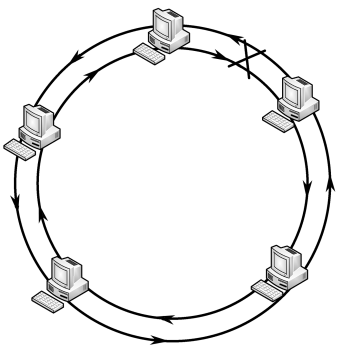


图 1-6　双通道环

1.3.4　树型拓扑结构

树型拓扑结构（图 1-7、图 1-8）是从总线型拓扑结构或星型拓扑结构演变而来的。它有两种类型：一种是由多条总线连接而成的，传输介质不构成闭合环路而是分支电缆；另一种是星型拓扑的扩展，各节点按一定的层次连接起来，信息交换主要在上、下节点之间进行。树型拓扑结构的优点：结构灵活，易于扩展，维护方便，适用于汇集信息的应用要求。树型拓扑结构

的缺点：资源共享能力低下；传输延时长。

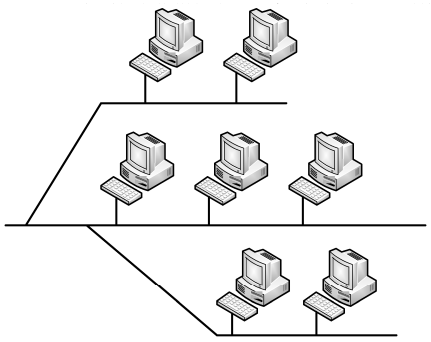


图 1-7　树型拓扑结构图（总线型演变）

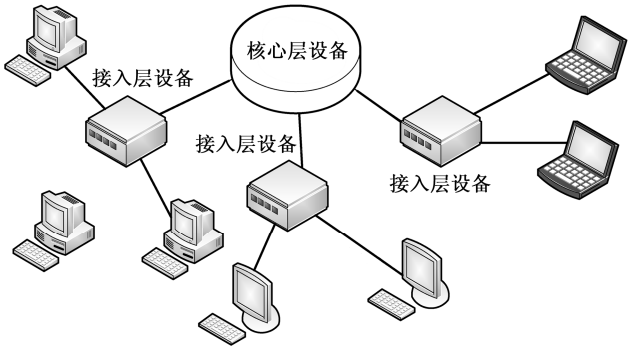


图 1-8　树型拓扑结构图（星型演变）

1.3.5　网状型拓扑结构

网状型拓扑结构（图 1-9）又称为分布式网络，是一种节点间可任意连接的结构。网络无中心主机，网络上的每个节点机都有两条或以上的线路与其他节点相连，从而增加了迂回通路。网状网络的通信子网是一个封闭式结构通信功能，分布在各个节点机上。

网状型拓扑结构的优点如下。

1）容易实现共享资源。

2）可改善线路的信息流量分配及均衡负载。

3）可选择最佳路径，传输延时小。

4）链路容错能力强，当某条链路中断时，可以通过其他链路进行信息交互。

网状型拓扑结构的缺点如下。

1）控制、管理和软件复杂导致安装维护困难。

2）布线工程量大，建设成本高。

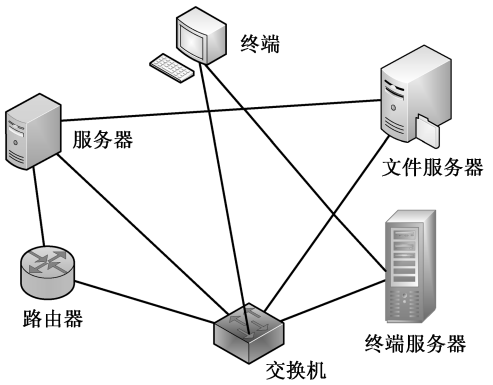


图 1-9　网状型拓扑结构图

1.3.6　混合型拓扑结构

混合型拓扑结构是由前面所讲的各种拓扑结构的网络结合在一起的网络结构，这样的拓扑结构更能满足较大网络拓展的需求，既可以利用现有拓扑结构的优点，又可以克服现有拓扑结构的缺点，如可以克服星型网络在传输距离上的局限，也可以克服总线型网络在连接用户数量的限制等。图 1-10 混合型拓扑结构的实例。

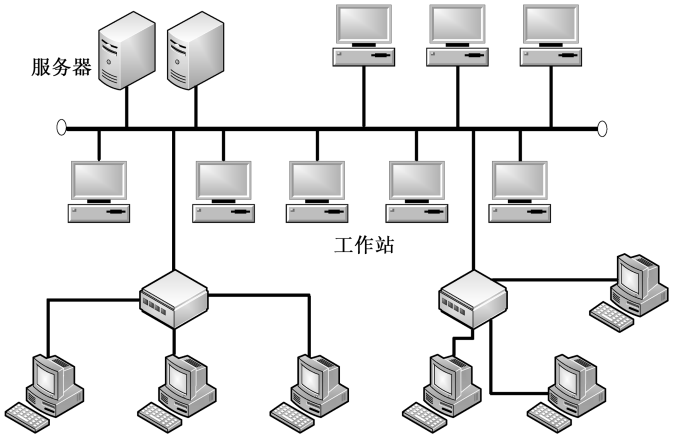


图 1-10　混合型拓扑结构图

混合型拓扑结构主要有以下几个方面的特点。

1）应用广泛，因为它可以利用各种拓扑结构的优点，又可以弥补现有拓扑结构的不足，满足了大公司组网的实际需求。

2）扩展灵活，其主要继承了星型拓扑结构的优点，但因为息传送方式，所以在总线长度和节点数量上也会受到限制，不过在局域网中不存在太大的问题。

3）网络速率会随着用户的增多而下降。

4）较难维护，这主要受到总线型网络拓扑结构的制约，如果总线断，则整个网络也就无法正常工作，但是如果分支网段出了故障，则仍不影响整个网络的正常运作。

5）速度较快，因为其骨干网采用高速的同轴电缆或光缆，所以整个网络在速度上不受太多的限制。

1.3.7　蜂窝型拓扑结构

蜂窝型拓扑结构主要用于移动通信网络中，这种结构是在 20 世纪 70 年代由美国贝尔实验室提出来的。其核心思想：将单个大功率基站覆盖的区域划分为多个六边形小区，每个小区使用小功率基站，每个小区分配整个系统可用信道中的一小部分，相邻基站则分配另外一些不同的信道，在相距较远的小区采用相同的频率。通过这种方法实现了频率复用，小区之间虽有干扰，但是可以控制它们之间的干扰在可接受的水平。因此，该结构可提高频谱利用率，减少相互干扰，增加系统容量。

综上所述，各种网络拓扑结构各有其优缺点，在网络规划时需要根据设计目标合理选择拓扑结构，同时要考虑网络长远规划、成本及操作维护等。对于单一业务网络，使用比较简单的拓扑结构即可实现，但是现在通信网承担的业务种类繁多，某一种拓扑结构已经无法满足其需求，需要各种拓扑结构的结合。

1.4 网络体系结构

数据通信网体系结构依照功能分层原理，采用高度结构化的设计方法，对整个通信过程的各个环节制定规划或约定，使网络中的每个节点都遵守一套合理而严谨的结构化管理体系进行通信。下面主要阐述协议、体系、结构的概念和分层的思想等内容。

1.4.1 网络协议与网络体系结构

1．网络协议的概念

在数据通信网中为数据交换而建立的规则、标准及约定的集合称为网络协议（Network Protocol），简称协议。数据通信网络中，处于不同地理位置的实体要进行通信，其上的两个进程相互通信，需要通过交换信息来协调它们的动作达到同步，而信息的交换必须按照预先共同约定好的规则进行。

网络协议有以下 3 个要素。

- 语法（Syntax）：指数据与控制信息的结构或格式。
- 语义（Semantics）：对通信过程进行规定，规定何时发出何种控制信息、完成何种动作及做出何种响应。
- 时序（Timing）：是对事件实现顺序的详细说明，指出事件的顺序及速度匹配。

“语法”规定通信双方“如何讲”，“语义”规定通信双方“讲什么”，“时序”规定通信双方的“应答关系”。

2．网络协议的特点

数据通信网络协议采用了分层的思想，网络中相互通信的功能被分为多个层次，每个层次之内还可以再分为若干子层。每个功能层都有自己的协议规范，而且彼此之间相互独立，不影响其他层次的通信。层与层之间接口应清楚，相互传递的信息要尽可能少。

3．网络体系结构

随着数据通信网规模的增大和复杂化，用来约定通信过程的网络协议也越来越复杂。网络开发人员和用户意识到需要一种总体的规划来指导网络及其应用的开发。网络设计人员能够根据技术进步，随时更改网络拓扑结构、网络提供的服务和其他特征，而不影响当前用户。网络用户也不应该看到网络每天发生的变化。网络应该能够接受各种不同的设备接入网络中，并且能够协同处理任务，由此产生了网络体系结构。

网络体系结构是设计和实现通信网络的原则，是网络层次模型和各层协议的集合。它是根据用户需求的规划和说明，包括网络具有的功能、数据格式和过程等，但是不规定网络如何实现。为了减少网络体系结构设计和实现的复杂程度，通常按照结构化方法进行设计，即将协议按功能分成若干层，每层完成一定的功能，并对其上层提供支持；每一层建立在下一层之上，

即每一层功能的实现以其下层提供的服务为基础；每一层都对上一层屏蔽如何实现协议和服务的具体细节。这样，不同厂商、不同型号、不同性能的计算机都可以连接到网络中，只要它们遵守相同的协议即可。这种方式使得网络体系结构与具体的物理实现无关。

结构化网络体系结构的关键在于合理划分层次，确定每一层的服务功能和不同层之间的接口。网络中各层需要具有的功能主要有差错控制、流量控制、分段与重装、复用与分用、寻址、连接的建立与释放等，根据各层所在的位置，合理选择各层的功能，使得各层的功能易于标准化。不同系统分成具有相同的层次，对等层次具有相同的功能。分层的层数也要适中，分层太多，会造成系统处理时间的增加，系统开销的增加，影响系统的传输速率和效率；层次太少，会造成每层功能过于复杂，相邻层接口不易确定，最终造成协议不稳定。

在同一系统中，相邻两层交换信息的连接点称为接口。接口定义了下层向上层提供的服务和有关使用这些服务对应的操作和响应。在同一系统中，上层向下层请求服务时，两者之间交换的信息称为服务原语。为了保证系统的可靠性，在执行过程中，服务原语不可中断或并发。

由此可以看出，结构化网络体系结构各层之间都是相互独立的，结构上可以分开，各层可以自由选择最合适的技术，而且当任何一层发生变化时，只要层间接口不变，则相邻的两层均不受影响。这种灵活的结构使得整个系统易于实现和维护。同时由于每一层的功能和所能提供的服务比较明确，可以促进标准化工作的进展。

首先提出网络体系结构概念的是美国的 IBM 公司。IBM 公司于 1974 年提出了系统网络体系结构（System Network Architecture，SNA），这个网络体系结构采用分层结构化方法。1978 年，ISO 提出了 OSI 模型，该协议试图使全世界的计算机通信网络都能够方便地进行互连和数据传输。

1.4.2　OSI 模型

国际标准化组织在 20 世纪 80 年代提出了 OSI 模型（也称 OSI/RM 模型），并定义了网络互连的 7 层框架，由低到高分别为物理层（Physical Layer）、数据链路层（Data Link Layer）、网络层（Network Layer）、传输层（Transport Layer）、会话层（Session Layer）、表示层（Presentation Layer）和应用层（Application Layer）。其中 1～3 层为低层，组成通信子网，负责创建网络通信连接的链路；5～7 层为高层，组成资源子网，负责端到端的数据通信；第 4 层（传输层）承接上下两部分，起连接作用。分层结构如图 1-11 所示。

1．OSI 模型的主要特性

- 它定义了异构系统互连的标准框架，采用了结构化的分层体系，规定了各层的功能等信息，不规定具体实现。
- 不同主机上的相同层为对等层；不同主机的对等层之间的虚通信必须遵循相应层的协议，若应用层协议、数据链路层协议等。
- 相邻层间的接口定义了原语操作和低层向上层提供的服务。

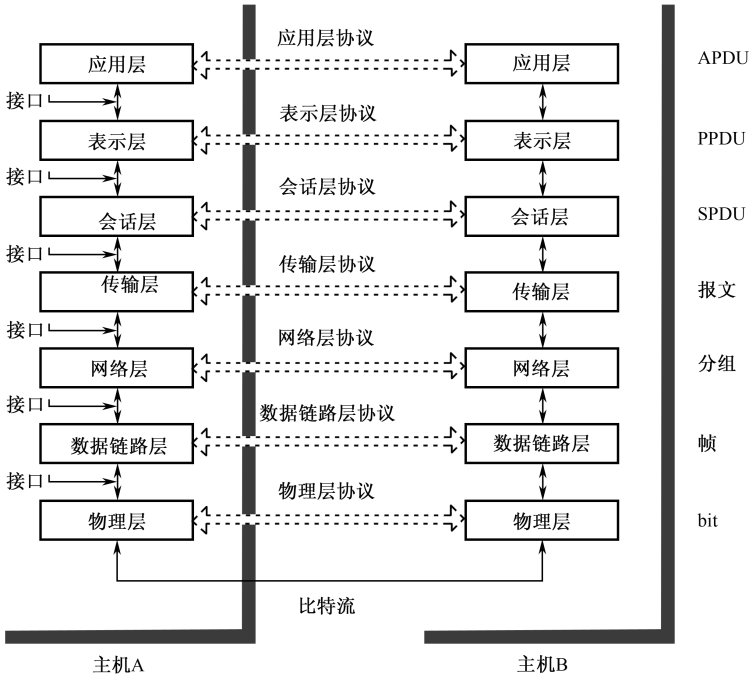


图 1-11　OSI 模型

- 所提供的公共服务是面向连接或无连接的数据通信服务。
- 直接的数据传送仅在底层实现
- 每层完成所定义的功能，修改本层的功能并不影响其他层。

2．OSI 模型的信息传导

不同主机的对等层之间的通信称为虚通信，这是因为层间交互并不是两对等层直接交互，其实际路径是每层都向各自的下层请求服务，直到底层，即物理层，以比特流（0、1 代码）传送到另一主机的物理层，再依次向上传。总的来说，双方的通信是在对等层次上进行的，不能在不对称层次上进行通信。

3．OSI 模型各层的功能

（1）物理层

物理层在 OSI 模型的最底层，它直接与物理信道相连，作为系统和通信介质的接口，实现两个系统之间的物理连接。其主要功能：为通信的网络节点之间建立、管理和释放数据电路的物理连接，并确保在通信信道上传输可识别的透明比特流信号。物理层所描述的传输媒体特性有以下 4 个。

- 机械特性：规定物理连接时接口所用的接线器的规格，如形状、尺寸、引线数目和排列方式等。
- 电气特性：规定物理信道上传输信号的参数，如信号电平、阻抗匹配和传输性能等。
- 功能特性：规定接口连线的功能，并说明某条连线上出现的某一电平的电压所表示的意义。

- 规程特性：规定了使用接口线实现数据传输的操作过程，也就是建立、维护和释放物理连接。

在这一层，数据传输的单位称为比特（bit）。

具有物理层功能的协议有 CCIYY V.24、EIA RS-443、EIA RS-232-C 和 ISO-2593 等。

物理层的设备有 RJ-45、各种线缆及接线设备。

（2）数据链路层

数据链路层在 OSI 模型的第二层，它接收物理层提供的比特流服务，在相邻网络实体之间建立、维持和释放数据链路的连接。其主要功能：在发送端和接收端之间进行可靠的、透明的数据传输，对链路连接进行建立、拆除和分离，以及为网络层提供服务。数据链路层的功能还包括链路管理、帧的封拆、帧同步、差错控制和流量控制等。

- 链路管理：当有数据传输时，为节点之间建立可靠的数据传输提供必要的链路建立、维持和释放机制。
- 帧的封拆：在发送端，将网络层传下来的分组按照帧的大小进行分割，然后在首部和尾部添加控制信息，即在分组上添加帧头和帧尾，形成数据链路层可以传输的帧单元。在接收端，数据链路层将接收到的帧剥去首尾，重新拆装成网络层的分组。
- 帧同步：发送端和接收端保持同步，以便保证接收端能够从收到的数据中准确地分出帧的开始和结束。
- 差错控制：数据信息在物理层传输时会受到各种因素的影响而产生差错，为了保证物理层数据的可靠传输，数据链路层需要对接收到的数据帧进行差错校验，并向发送方反馈有关接收情况的信息。
- 流量控制：为了防止由于发送端和接收端因数据处理能力不匹配而造成的数据丢失，需要对发送端发送的数据速率进行控制，使得接收端能够及时处理接收到的数据。

在这一层，数据传输的单位称为帧（Frame）。

具有数据链路层功能的协议有 ATM、IEEE 802.2、帧中继和 HDLC 等。

数据链路层的设备有集线器和交换机等。

（3）网络层

网络层在 OSI 模型的第三层，它负责控制通信子网的操作。其主要功能：选择合适的网间路由和交换节点，并为分组中继，网络互连、流量控制、拥塞控制等，将数据从源节点经过若干中间节点传送至目的节点，实现两个节点之间的数据透明、可靠、传输。

- 逻辑地址寻址：同一网络内部不同的节点通过数据链路层的物理地址来进行通信；不同网络之间的节点通过网络层的地址来进行通信。当网络层收到传输层的报文时，将报文转换为数据包，为保证数据包在网络中的传输，需要在数据包中加入网络层地址。网络层寻址与数据链路层无关。
- 路由功能：随着网络规模的增大，源节点和目的节点之间存在多条可选的路径，需要根据一定的原则和算法在可选路径中选择一条最佳路径。
- 流量控制：数据链路层流量控制在两个相邻节点间进行，网络层流量控制在源节点和目的节点通信过程中进行。

- 拥塞控制：网络中出现过多的数据包时会引起网络性能的下降，称为网络拥塞。为了避免这种现象，需要对网络进行拥塞控制。

在这一层，数据传输的单位是数据包（Packet）。

具有网络层功能的协议有 IP、IPX、RIP、OSPF 等。

网络层的设备有路由器和三层交换机等。

（4）传输层

传输层在 OSI 模型的第四层，是资源子网和通信子网的接口。其主要功能：向用户提供可靠的端到端的透明的、可靠的数据传输服务。OSI 模型中的物理层、数据链路层和网络层均属于通信子网，完成有关通信的功能；传输层、会话层、表示层和应用层均属于资源子网，完成数据处理功能。传输层在 OSI 模型的中间，起承上启下的作用，如图 1-12 所示。

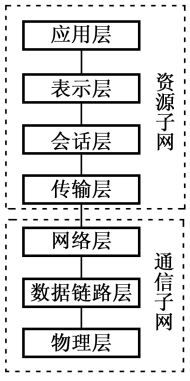


图 1-12　传输层在 OSI 模型中的地位

在这一层，数据传输的单位是数据段（Segments）或数据报（Datagram）。

具有传输层功能的协议有 TCP、UDP、SPX 和 NetBIOS 等。

（5）会话层

会话层在 OSI 模型的第五层，也可称为会晤层或对话层。其主要功能：在传输层所提供的服务的基础上为两主机的用户进程建立会话连接。会话层不参与具体的传输，它提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。例如，服务器验证用户登录便是由会话层完成的。会话层提供的服务可使应用建立和维持会话，并能使会话获得同步。会话层使用校验点，可使通信会话在通信失效时从校验点继续恢复通信，下载文件时使用的“断点续传”就工作在会话层。

（6）表示层

表示层在 OSI 模型的第六层，向上为应用层服务，向下接收会话层的服务，它关心所传输信息的语法和语义。该层的主要功能：通过数据格式的变换、数据加密与解密、数据压缩与恢复等步骤将数据表示为用户所需要的形式。

具有表示层功能的协议有 HTTP/HTML、FTP、Telnet 和 ASN.1 等。

（7）应用层

应用层在 OSI 模型的第七层，也是最高层，直接面向用户，是计算机网络与最终用户的界面，负责两个应用进程之间的通信，为网络用户之间的通信提供专用程序。应用层识别并证实目的通信方的可用性，使协同工作的应用程序之间进行同步，建立传输错误纠正和数据完整性控制的协议。它还判断是否为所需的通信过程留有足够的资源。

具有应用层功能的协议有 FTP、SMTP、DNS 和 POP 等。

1.4.3　TCP/IP 模型

由于 OSI 模型的标准进展缓慢，美国政府的高级研究计划署开发了一组称为 TCP/IP（Transmission Control Protocol/Internet Protocol）的协议，用于支持网络互连。美国国防部最终

要求在它的所有计算机和网络中使用TCP/IP，这为安装协议的设备自动提供了很强的基本功能，并为软件和设备供应商提供了很大的市场机会。基于以上两个原因，TCP/IP 最终成为 Internet 体系结构和协议的基础。

TCP/IP 体系是目前被广泛使用的网络协议，几乎所有的厂商和操作系统支持它。TCP/IP 也是 Internet 的基础协议。

TCP/IP 模型有 4 层，分别为应用层、传输层、网络层和接口层。OSI 模型有 7 层，TCP/IP 模型和 OSI 模型的对应关系如图 1-13 所示。

1．TCP/IP 模型的主要特性

- 开放的免费协议标准，并且独立于计算机的硬件和操作系统。
- 可以越层服务，通信效率较高。
- 协议规定每层功能，不规定物理实现方法。
- 采用统一的 IP 地址，可以适用于不同网络的互连互通。
- TCP/IP 应用层的功能包含了会话层和表示层的功能，接口层包括了 OSI 模型的物理层和数据链路层。

图 1-14 所示为TCP/IP分层工作原理示意图。

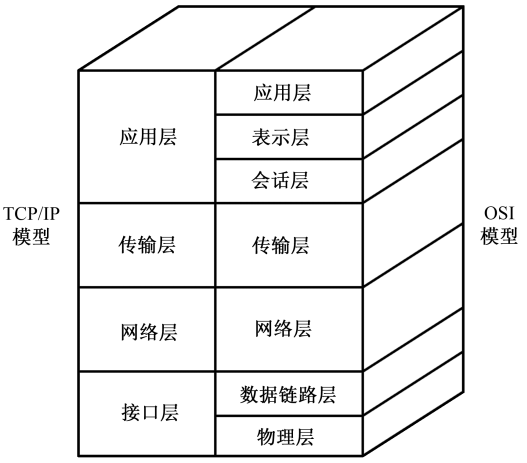


图 1-13　TCP/IP 模型和 OSI 模型的对应关系

2．TCP/IP 模型各层的功能

（1）接口层

接口层位于 TCP/IP 模型的最底层，对应着 OSI 模型的物理层和数据链路层。接口层负责网络 IP 数据的发送和接收。接口层只是一个访问接口，没有定义具体的网络接口协议，与物理传输介质无关，增加了设计的灵活性，可以适应各种物理网络。

（2）网络层

网络层位于 TCP/IP 模型的第二层，又被称为 IP 层，它相当于 OSI 模型的网络层。它的功能是处理传输层的分组，将源节点的数据传送到网络中的任何目的节点。源节点和目的节点可以在同一网络中，也可以在不同的网络中。网络层只是尽力地将 IP 数据分发到目的节点，对可靠性没有任何保证。

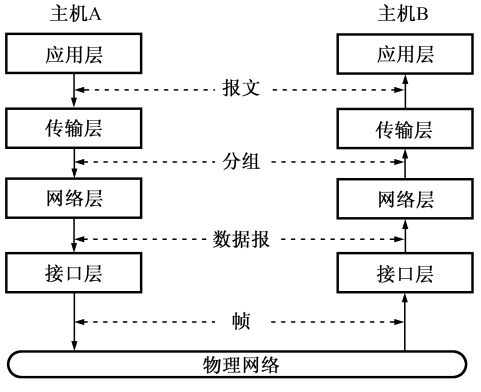


图 1-14　TCP/IP 分层工作原理

（3）传输层

传输层位于 TCP/IP 模型的第三层，主要负责端到端的可靠通信。为了提供可靠通信，传输层定义了 3 个协议： TCP、用户数据报协议（User Datagram Protocol，UDP）和差错和控制报文协议（Internet Control Message Protocol，ICMP）。TCP 是面向连接的协议，具有差错校验、重发和顺序控制等功能，可以保证数据的可靠传输。UDP 是面向无连接的协议，没有额外开销，效率较高。ICMP 主要具有错误信息发送和流量控制功能。

（4）应用层

应用层位于 TCP/IP 模型的最高层，主要用来提供各种应用服务。TCP/IP 提供的应用服务有远程登录协议（Telnet）、简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）、域名服务（DNS）、文件传输协议（FTP）、超文本传输协议（HTTP）等。

小结

数据通信网是当今信息化社会中必不可少的一部分，而且随着网络技术的迅猛发展，数据通信网技术的内容不断丰富，已经形成了一个比较完整的理论和技术体系。

本章的目的主要是使读者掌握数据通信的基本知识，了解其发展、分类、结构等。本章重点为数据通信网的分类、网络拓扑结构的种类和特点及 OSI 模型和 TCP/IP 模型的分层及各层作用。

参考文献

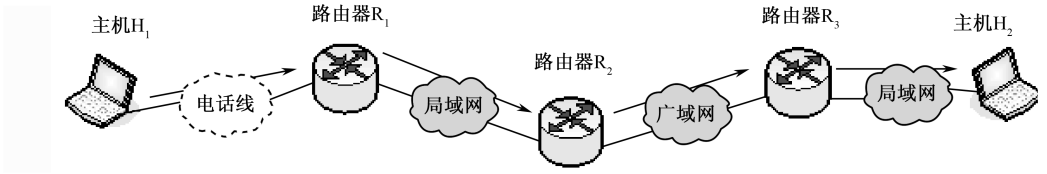
[1] 韩希义．计算机网络技术基础[M]. 北京：机械工业出版社，2010.
[2] 胡远萍．计算机网络技术及应用[M]. 北京：高等教育出版社，2009.
[3] 胡道元．计算机网络[M]. 2 版. 北京：清华大学出版社，2009.
[4] 向隅．计算机网络基础[M]. 北京：清华大学出版社，北京交通大学出版社，2009.
[5] 裴有柱，张扬．计算机网络技术与实训教程[M]. 北京：机械工业出版社，2010.
[6] 陈代武．计算机网络技术[M]. 北京：北京大学出版社，2009.
[7] Stanford H.Rowe，Marsha L.Schuh. Computer Networking [M]. America, 2006.
[8] Dave Miler，数据通信与网络. 北京：清华大学出版社，2007.
[9] 朱辉，等．数据通信网络技术[M]. 西安：西安电子科技大学出版社，2012.
[10] 黎连业，陈俊良，黎萍．计算机网络系统集成与方案实例[M]. 2 版. 北京：机械工业出版社，2007.
[11] 谢希仁．[M]. 北京：计算机网络，电子工业出版社，2010.
[12] 刘衍衍，王健．数据通信[M]. 北京：机械工业出版社，2013.
[13] 李斯伟，胡成伟．数据通信技术[M]. 北京：人民邮电出版社，2011.
[14] 毛京丽，等．数据通信原理[M]. 北京：北京邮电大学出版社，2002.
[15] 达新宇，等．数据通信原理与技术[M]．2 版. 北京：电子工业出版社，2010.

第 2 章

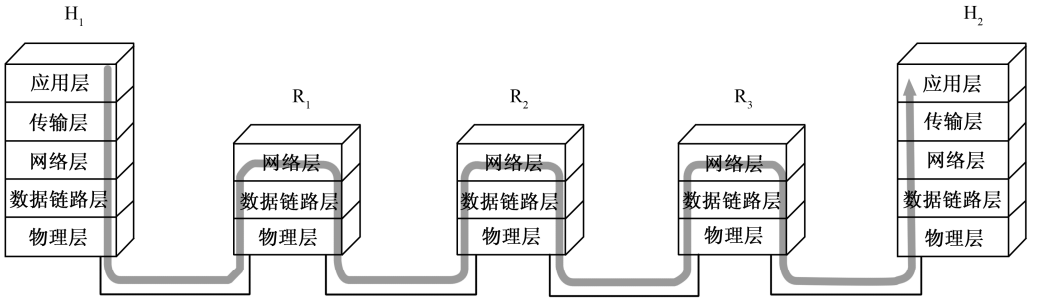
数据链路层和网络层

2.1 数据链路层

数据链路层是 OSI 参考模型中的第二层，介于物理层和网络层之间。数据链路层是在物理层提供的服务基础上向网络层提供服务，其最基本的服务是将源机网络层送来的数据可靠地传输到相邻节点的目标机网络层。为达到这一目的，数据链路必须具备一系列相应的功能，主要有：如何将数据组合成数据块，在数据链路层中称这种数据块为帧，帧是数据链路层的传送单位；如何控制帧在物理信道上的传输，包括如何处理传输差错，如何调节发送速率以使与接收方相匹配；以及在两个网络实体之间提供数据链路通路的建立、维持和释放的管理。图 2-1 展现了两个主机通过互联网进行通信时数据链路层所处的地位。



（a）主机H_1向主机H_2发送数据



（b）从层次上看数据流动

图 2-1　数据链路层的地位

图 2-1（a）表示主机 H_1 通过电话线上网，中间经过 3 个路由器（R_1、R_2 和 R_3）连接到远程主机 H_2。所经过的网络可以有多种，如电话网、局域网和广域网。当主机 H_1 向 H_2 发送数据时，从协议的层次上看，数据的流动如图 2-1（b）所示。主机 H_1 和 H_2 都有完整的 5 层协议栈，但路由器在转发分组时使用的协议只有下面的 3 层。数据进入路由器后要先从物理层上到网络层，在转发表中找到下一帧的地址后，再下到物理层转发出去，因此数据从主机 H_1 传送到主机 H_2 需要在路径中的各节点协议栈向上和向下流动多次。

然而当专门研究数据链路层的问题时，在许多情况下我们可以只关心在协议栈中水平方向的各数据链路层，当主机 H_1 向主机 H_2 发送数据时，我们可以想象数据就是在数据链路层从左向右沿水平方向传送，如图 2-2 中从左到右的粗箭头所示，即通过以下这样的链路传送：H_1 的数据链路层→R_1 的数据链路层→R_2 的数据链路层→R_3 的数据链路层→H_2 的数据链路层。

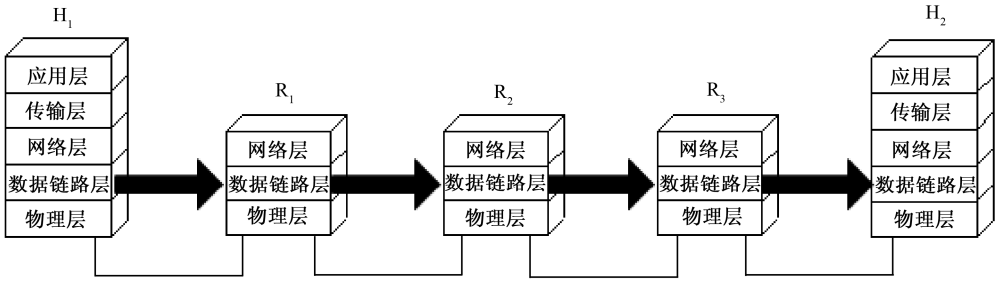


图 2-2　只考虑数据在数据链路层的流动

图 2-2 指出，从数据链路层来看，H1 到 H2 的通信可以看成由 4 段不同的数据链路层通信组成，即 H_1→R_1、R_1→R_2、R_2→R_3 和 R_3→H_2。这 4 段不同的数据链路层可能采用不同的数据链路层协议。

2.1.1　数据链路和帧

在这里要明确一下，“链路”和“数据链路”并不是一回事。所谓链路（Link）就是从一个节点到相邻节点的一段物理线路，而中间没有任何其他的交换点。在进行数据通信时，两个计算机之间的通信路径往往要经过许多段这样的链路，可见链路只是一条路径的组成部分。

数据链路（Data Link）则是另一个概念。这是因为当需要在一条线路上传送数据时，除了必须有一条物理线路外，还必须有一些必要的通信协议来控制这些数据的传输，若把实现这些协议的硬件和软件加到链路上，就构成了数据链路。现在最常用的方法是使用网络适配器（如拨号上网时，使用拨号适配器；通过以太网上网时，使用局域网适配器）来实现这些协议的硬件和软件的添加，一般的适配器包括了数据链路层和物理层这两层的功能。也有人采用另外的术语，即把链路分为物理链路和逻辑链路。物理链路就是上面所说的链路，而逻辑链路就是上面的数据链路，是物理链路加上必要的通信协议。早期的数据通信协议曾称为通信规程（Procedure），因此在数据链路层，规程和协议是同义语。下面介绍信道的数据链路层的协议数

据单元——帧。

数据链路层把网络层传来的数据构成帧发送到链路上，并把接收到的帧中的数据取出上交给网络层。在 Internet 中，网络层协议数据单元就是 IP 数据报（或简称数据报、分组或包）。为了能把主要的精力放在信道的数据链路层协议上，可以采用如图 2-3(a)所示的 3 层模型。在这种 3 层模型中，不管在哪一段链路上的通信（主机和路由器之间或两个路由器之间），都看成节点和节点的通信（如图 2-3 中的节点 A 和 B），而每个节点只有下 3 层——网络层、数据链路层和物理层。

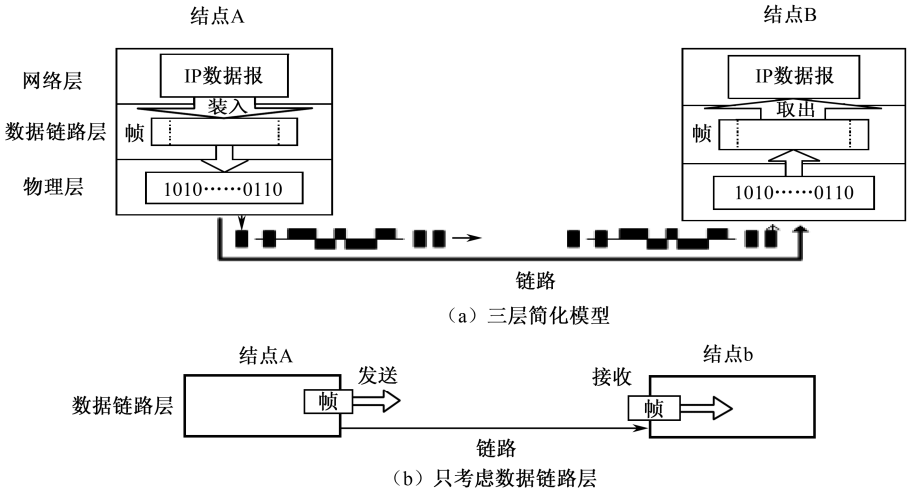


图 2-3　使用信道的数据链路层

信道的数据链路层在进行通信时的主要步骤如下。

1）节点 A 的数据链路层把网络层传来的 IP 数据报添加首部和尾部封装成帧。

2）节点 A 把封装好的帧发送给节点 B 的数据链路层。

3）若节点 B 的数据链路层收到的帧无差错，则从收到的帧中提取 IP 数据报上交给上面的网络层，否则丢弃这个帧。

数据链路层不必考虑物理层如何实现比特传输的细节，我们甚至还可以更简单地设想好像是沿着两个数据链路层之间的水平方向把帧直接发送到对方，如图 2-3（b）所示。

2.1.2　3 个基本问题

数据链路层协议有许多种，但有 3 个基本问题则是共同的。这 3 个基本问题是封装成帧、透明传输和差错检测。下面分别讨论这 3 个问题。

1．封装成帧

封装成帧（Framing）就是在一段数据的前后分别添加首部和尾部，这样就构成了一个帧。接收端在收到物理层上交的比特流后，就能根据首部的标记，从收到的比特流中识别帧的开始

和结束。图 2-4 表示为用帧首部和帧尾部封装成帧的一般概念。众所周知，分组交换的一个重要概念是：所有在 Internet 上传送的数据都以 IP 数据报为传送单位。网络层的 IP 数据报传送到数据链路层就成为帧的数据部分，在帧的数据部分的前面和后面分别添加首部和尾部，就构成了一个完整的帧。因此，帧长等于数据部分的长度加上帧首部和尾部的长度，而首部和尾部的一个重要作用就是进行帧定界（确定帧的界限）。此外，首部和尾部还包括许多必要的控制信息，各种数据链路层协议都要对帧首部和帧尾部的格式有明确的规定。显然，为了提高帧的传输效率，应当使帧的数据部分长度尽可能大于首部和尾部的长度。但是，每一种链路层协议都规定了帧的数据部分的长度上限——最大传送单元（Maximum Transmission Unit，MTU）。图 2-4 给出了帧首部和尾部的位置，以及帧的数据部分与 MTU 的关系。

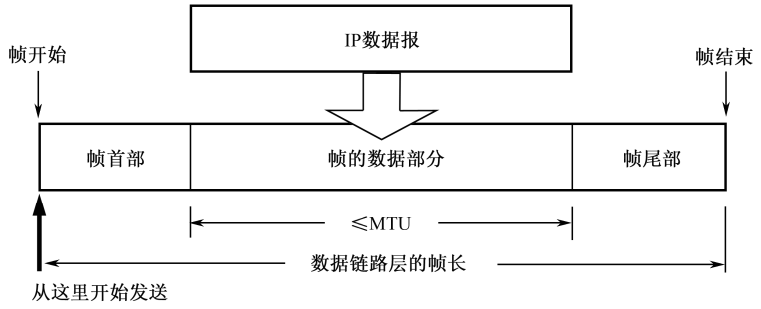


图 2-4　用帧首部和尾部封装成帧

当数据是由可打印的 ASCII 码组成的文本文件时，帧定界可以使用特殊的帧定界符。我们知道，ASCII 码是 7 位编码，一共可组合成 128 个不同的 ASCII 码，其中可打印的有 95 个，而不可打印的控制字符有 33 个，图 2-5 的例子可说明帧定界的概念。控制字符 SOH（Start Of Header）放在一帧的最前面，表示帧开始，另一个控制字符 EOT（End Of Transmission）表示帧结束。请注意，SOH 和 EOT 都是控制字符的名称，它们的十六进制编码分别是 01（二进制为 00000001）和 04（二进制为 00000100），SOH（或 EOT）并不是 S、O、H（或 E、O、T）这 3 个字符。

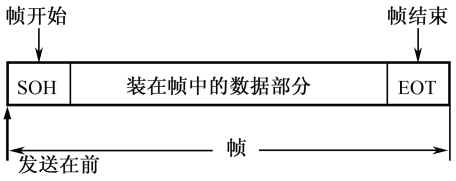


图 2-5　用控制字符进行帧定界的方法举例

当数据在传输中出现差错时，帧定界符的作用更加明显。假定发送端在尚未发送一帧时突然出现故障，中断了发送，但随后很快又恢复正常，于是重新开始发送刚才未发送完的帧。由于使用了帧定界符，在接收端就知道前面的数据是个不完整的帧（只有首部开始符 SOH 而没有传输结束符 EOT），必须丢弃，而后面收到的数据有明显的帧定界符（SOH 和 EOT），因此这是一个完整的帧，应当收下。

2．透明传输

由于帧开始和结束的标记使用专门的控制字符，因此所传输数据中的任何 8 bit 的组合一定不允许和用于帧定界符的控制字符的比特编码一样，否则就会出现帧定界符的错误。当传送的

帧是用文本文件组成的帧时（文本文件中的字符都是从键盘输入的），其数据部分显然不会出现像 SOH 或 EOT 这样的帧定界符控制字符。可见不管从键盘上输入什么字符，都可以放在这样的帧中传输过去，因此这样的传输就是透明传输。

但当数据部分是非 ASCII 码的文本文件时（如二进制代码的计算机程序或图像等），情况就会不同。如果数据中的某字节的二进制代码恰好和 SOH 或 EOT 这种控制字符一样（如图 2-6 所示），数据链路层就会错误地“找到帧的边界”，把部分帧收下（误认为是个完整的帧），而把剩下的那部分数据丢弃（这部分找不到帧定界符控制字符 SOH）。

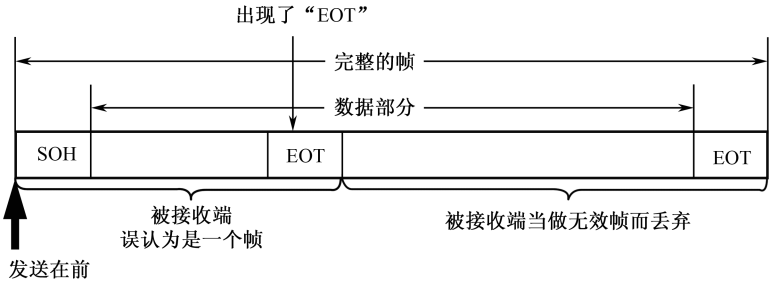


图 2-6　数据部分恰好出现了与 EOT 一样的代码

如图 2-6 所示的帧传输显然就不是透明传输，因为当遇到数据中碰巧出现字符“EOT”时就传过去了。数据中“EOT”将被接收端错误地解释为“传输结束”的控制字符，而其后面的数据因找不到“SOH”被接收端当作无效帧而丢弃，但实际上在数据中出现的字符“EOT”并非控制字符，而仅仅是二进制数据 00000100。

为了解决透明传输问题，就必须设法使数据中可能出现的控制字符“SOH”和“EOT”在接收端不被解释为控制字符。具体的方法是：发送端的数据链路层在数据中出现的控制字符“SOH”或“EOT”的前面插入一个转义字符“ESC”（其十六进制编码是 1B），而在接收端的数据链路层在将数据送往网络层之前删除这个转义字符，这种方法称为字节填充（Byte Stuffing）或字符填充（Character Stuffing）。如果转义字符也出现在数据当中，那么解决方法仍然是在转义字符的前面插入一个转义字符，当接收端收到连续的两个转义字符时，就删除前面的一个。图 2-7 表示为用字节填充法解决透明传输的问题。

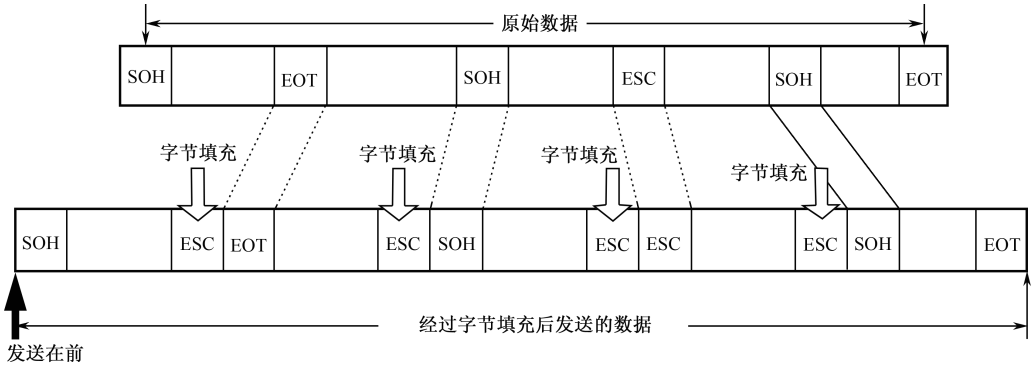


图 2-7　用字节填充法解决透明传输的问题

3．差错检测

现实的通信链路都不会是理想的，这就是说，比特在传输过程中可能会产生差错：1 可能会变成 0，而 0 也可能变成 1，这称为比特差错。比特差错是传输差错中的一种，本节所说的“差错”，如无特殊说明，皆指比特差错。在一段时间内，传输错误的比特占所传输比特总数的比例为误码率（Bit Error Rate，BER）。例如，误码率为 10^{-10} 时，表示平均每传 10^{10} 个比特，就会出现一个比特的差错。误码率与信噪比有很大的关系，如果设法提高信噪比，就可以使误码率减小，实际的通信链路并非理想，它不可能使误码率下降到零。因此，为了保证数据传输的可靠性，在计算机网络传输数据时，必须采用各种差错检测措施，目前在数据链路层广泛使用了循环冗余检验（Cyclic Redundancy Check，CRC）的检错技术。下面我们通过一个简单的例子来说明循环冗余检验的原理。

在发送端，先把数据划分为组，假定每组有 k 比特，现假设待传送的数据 M=101001（k=6）。循环冗余检验就是在数据 M 的后面添加供差错检测用的 n 位冗余码，然后构成一个帧发送出去，一共发送（k+n）位。在所要发送的数据后面增加 n 位的冗余码，虽然增大了数据传输的开销，但可以进行差错检测。当传输可能出现差错时，付出这种代价往往是很值得的。

这 n 位冗余码可通过以下方法得出。用二进制的模 2 进行 $2^n \times M$ 的运算，这相当于在 M 后面添加 n 个 0。得到的（k+n）位的数除以收发双方事先商定的长度为（n+1）位的除数 P，得出商是 Q 而余数是 R（n 位，比 P 少一位），关于除数 P 下面还要介绍。在图 2-8 所示的例子中，M=101001（k=6），假定除数 P=1101（n=3）。经过 2 除法运算后的结果是：商 Q=110101（这个商并没有用处），而余数 R=001，这个余数 R 就作为冗余码拼接在数据 M 的后面发送出去。这种为了进行检错而添加的冗余码常称为帧检验序列（Frame Check Sequence，FCS）。因此加上帧检验序列后发送的帧是 101001001（$2^n M$ +FCS），共有（k+n）位。

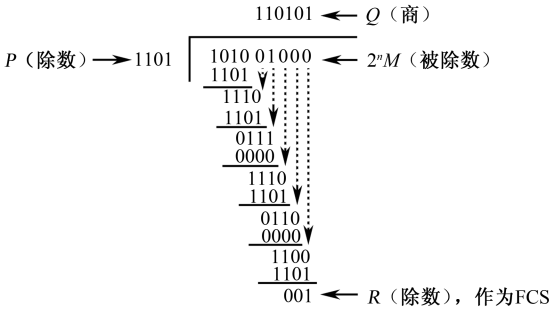


图 2-8　说明循环冗余检验原理的例子

循环冗余检验和帧检验序列并不是同一个概念，循环冗余检验是一种检错方法，而帧检验序列是添加在数据后面的冗余码，在检错方法上可以选用循环冗余检验，但也可不选用循环冗余检验。

把接收端的数据以帧为单位进行循环冗余检验：把收到的每一帧都除以同样的除数 P（模 2 运算），然后查得到的余数 R。如果在传输过程中无差错，那么经过循环冗余检验后得出的余数 R 肯定是 0（读者可以自己进行验算，被除数现在是 101001001，而除数是 1101，看余数是否为 0），但如果出现误码，那么余数仍等于零的概率是非常小的（这可以通过不太复杂的概率计算得出）。

总之，在接收端对收到的每一帧经过循环冗余检验后，若得出的余数 R=0，则判定这个帧

没有差错，就接收。若余数 $R\neq0$，则判定这个帧有差错（但无法确定究竟是哪一位或哪几位出现了差错），就丢弃。

一种较为方便的方法是用多项式来表示循环冗余检验过程。在上面的例子中，多项式 $P(X)=X^3+X^2+1$ 表示上面的除数 $P=1101$（最高位对应于 X^3，最低位对应 X^0），多项式 $P(X)$称为生成多项式。现在广泛使用的生成多项式 $P(X)$有以下几种。

$$\text{CRC}-16=X^{16}+X^{15}+X^2+1$$

$$\text{CRC}-\text{CCITT}=X^{16}+X^{12}+X^5+1$$

$$\text{CRC}-32=X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$$

在数据链路层，发送端帧检验序列的生成和接收端的循环冗余检验都是用硬件完成的，处理很迅速，因此并不会延误数据的传输。

从以上讨论不难看出，如果我们在传送数据时不以帧为单位来传送，那么就无法加入冗余码以进行差错检验。因此，如果要在数据链路层进行差错检验，就必须把数据划分为帧，每一帧都加上冗余码，一帧接一帧地传送，然后在接收方逐帧进行差错检验。

最后再强调一下，在数据链路层若仅仅使用循环冗余检验差错检测技术，则只能做到对帧的无差错接受，即“凡是接收端数据链路层接受的帧，我们都能以非常接近于 1 的概率认为这个帧在传输过程没有产生差错”。接收端丢弃的帧虽然已曾收到，但最终还是因为有差错被丢弃，即没有被接受。以上所述可以近似地表示为（通常都是这样认为）：“凡是接收端数据链路层接受的帧均无差错”。

请注意，现在并没有要求数据链路层向网络层提供“可靠传输”的服务。所谓“可靠传输”就是数据链路层的发送端发送什么，在接收端就收到什么。传输差错可分为两大类：一类就是前面所说的最基本的比特差错；而另一类传输差错则更复杂，这就是收到帧且没有出现比特差错，但却出现了帧丢弃、帧重复或帧失序。例如，发送方连续传送 3 个帧，即[#1]-[#2]-[#3]，在接收端收到的却有可能出现下面的情况。

帧丢失：收到[#1]-[#3]（丢失[#2]）。

帧重复：收到[#1]-[#2] -[#2]-[#3]（收到两个[#2]）。

帧失序：收到[#1]-[#3]-[#2]（后发送的帧反而先到达了接收端，这与一般数据链路层的概念不一样）。

以上 3 种情况都属于“出现传输差错”，但都不是这些帧里有“比特差错”。帧丢失很容易理解，但出现帧重复和帧失序的情况则较为复杂，对这些问题我们现在不展开讨论。总之，我们应当明确，“无比特差错”与“无传输差错”并不是同样的概念，在数据链路层使用循环冗余检验，能够实现无比特差错的传输，但这还不是可靠传输。

我们知道，OSI 的观点是必须让数据链路层可靠传输。因此在循环冗余检验的基础上，增加了帧编号、确认和重传机制。收到正确的帧就要向发送端发送确认，发送端在一定的期限内若没有收到对方的确认，就认为出现了差错，因而就进行重传，直到收到对方的确认为止。这种方法曾经起到很好的作用，但现在的通信线路的质量已经大大提高，由通信链路层质量不好引起差错的概率已经大大降低。因此，Internet 广泛使用的数据链路层协议都不使用确认和重传机制，即不要求数据链路层向上提供可靠传输的服务（因为这要付出太高的代价）。如果在数

据链路层传输时出现了差错并且需要进行改正，那么改正差错的任务就由上层协议（如传输层的 TCP）来完成。实践证明，这样做可以提高通信效率。

2.1.3 数据链路控制

1．数据链路控制基本思想

一条可靠的数据链路应该满足以下两个条件。

1）传输的任何数据，既不会出现差错，又不会丢失。

2）不管发送端以多快的速率发送数据，接收端总能够来得及接收、处理并上交主机，也就是说，接收端有足够的接收缓存和处理速度。

但实际应用的数据链路并不能满足上述条件。第（1）个条件不满足，就必须进行差错控制（Error Control），第（2）个条件不满足就必须进行流量控制（Flow Control）。

差错控制使得链路传输出现差错时得到补救。主要有两种差错发送：一是帧丢失，即一个数据帧未能到达接收端；二是帧损坏，如其中有几位数据出错。差错控制的基本方式是反馈重传机制。

流量控制用来保证发送数据在任何情况下都不会“淹没”接收方的接收缓存（接收缓存溢出），从而不会丢失数据，而且还应该使传输达到理想的吞吐率。由接收端根据其接收缓存的状况来控制发送端的数据流量是流量控制的基本思想，实现流量控制的一个重要方法是滑动窗口（Sliding Window）机制。

2．数据链路控制的基本机制

（1）反馈重传机制

差错控制的常用方法是反馈重传机制，也称确认-重传机制，也是数据链路控制的一个基本机制。接收端对接收到的数据进行差错检验后，以某种方式向发送端反馈差错状况，称为确认（Acknowled-gement），发送端根据确认信息对出现传输差错的帧进行重传。

反馈重传机制包括以下两步。

1）接收端反馈确认信息。确认方式包括多种。

① 正确认或肯定确认（Positive Acknowledgement）。接收端收到一个经检验无错的帧后，返回一个正确认。正确认简称确认，记为 ACK（ACKnowledgement）。

② 累计确认（Cumulative Acknowledgement）。接收端收到多个连续且正确的数据帧以后，才只对最后一个帧发回一个 ACK，称为累计确认。

③ 捎带确认（Piggybacking）。在双向数据传输情况下，将确认信息放在自己数据的首部字段中捎带过去。累计确认和捎带确认都可提高传输效率。

④ 负确认（Negative ACKnowledgement，NAK）。接收端收到一个有差错的帧后，返回一个对此帧的 NAK。

2）发送端重传差错帧。常用的重传方式包括以下两种。

① 超时重传（Timeout Retransmission）。发送端在发送完一帧时即启动一个重传定时器，

若由它设定的重传时间到且未收到反馈的确认信息，则重传此帧，这是经常采用的重传方式。为了重传，发送端必须保存一个已发出的数据帧的副本。

② 负确认重传。发送端收到接收端对一个帧的 NAK，重传此帧。

反馈重传机制对出差错数据帧的重传是自动进行的，因此这种控制机制称为自动请求重传 ARQ。根据反馈重传方式的不同，可以分为停等 ARQ（Stop-and-wait ARQ）、回退-N ARQ（Go-back-N ARQ）和选择重传 ARQ（Selective Repeat ARQ）。

实际上，ARQ 既使用了反馈重传机制对传输过程进行差错控制，又同时使用了滑动窗口机制进行流量控制，从而保证数据链路层实现可靠地数据传输。

ARQ 是第二次世界大战期间发明的，目的是使无线通信提供可靠的字符传送。它非常简单，且对信道的噪声有很强的适应性。

（2）滑动窗口机制

滑动窗口是数据链路控制的一个基本机制，发送端和接收端分别设置发送窗口和接收窗口，在数据传输过程中，在接收端的控制下向前滑动，从而对数据传输流量进行控制。

发送窗口用来对发送端进行流量控制，落在窗口内的帧是可以连续发送的，其大小 W_T 指明在收到对方 ACK 之前发送端最多可以发送多少帧。

接收窗口控制哪些帧可以接收，只有到达帧的序号落在接收窗口之内时才可以被接收，否则将被丢弃。一般地，当接收端收到一个有序且无差错的帧后，接收窗口向前滑动，准备接收下一帧，并向发送端发出一个 ACK。

当发送端收到接收端的 ACK 后，发送窗口才能向前滑动，滑动的长度取决于接收端确认的序号。向前滑动后，又有新的待发帧落入发送窗口，可以被发送。

可见，接收端的 ACK 作为授权发送端发送数据的凭证，接收端可以根据自己的接受能力来控制确认的发送时机，从而实现对传输流量的控制。

下面以图 2-9 为例说明滑动窗口机制。假设发送序列用 3 bit 来编码，即发送序号可以有从 0～7 的 8 个不同的序号。又设发送窗口 W_T=5，发送端滑动窗口的工作过程如图 2-9 所示。

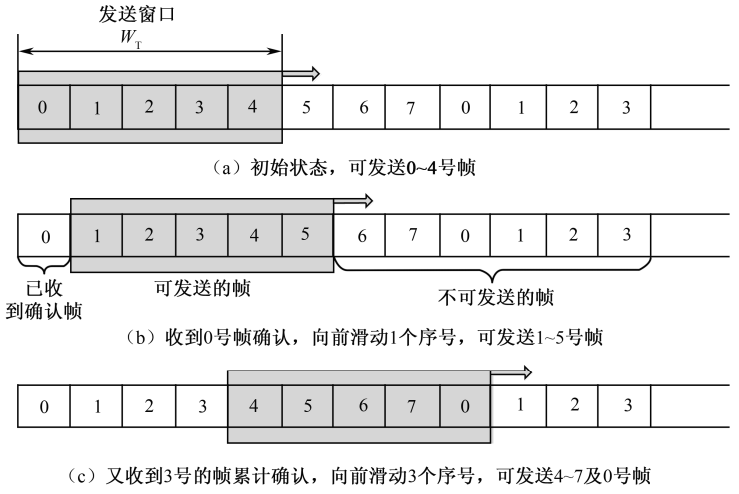

发送窗口
W_T
0 1 2 3 4 5 6 7 0 1 2 3
（a）初始状态，可发送0~4号帧
0 1 2 3 4 5 6 7 0 1 2 3
已收到确认帧
可发送的帧
不可发送的帧
（b）收到0号帧确认，向前滑动1个序号，可发送1~5号帧
0 1 2 3 4 5 6 7 0 1 2 3
（c）又收到3号的帧累计确认，向前滑动3个序号，可发送4~7及0号帧

图 2-9　发送窗口

1）初始状态如图 2-9（a）所示。发送窗口内共有从 0～4 的 5 个序号，这些帧现在可以连续发送，而 5 号及以后的帧不能发送。

2）收到了接收端对 0 号帧的确认 ACK1，发送窗口向前滑动 1 个序号，如图 2-9（b）所示。现在 5 号帧已经进入到发送窗口之内，可以发送。

3）在这以后，假设又收到对 3 号帧的累计确认 ACK4，说明接收端又正确地收到了 1～3 号帧，于是发送窗口又可再向前移动 3 个序号，那么 6 号、7 号和 0 号帧又进入发送窗口，它们也可以发送，如图 2-9（c）所示。

可见，图 2-9 中发送窗口左边的数据帧是已经发送并得到确认的帧；窗口内是可以发送的帧，包括已经发送但未得到确认的帧和尚未发送的帧；窗口的右边是不可以发送的数据帧。

滑动窗口机制中，为控制传输流量，可以设置合适大小的 W_T，一般不超过接收端接收缓存的大小，这样，发送的数据就不容易淹没接收的缓存。还可以使用可变滑动窗口，由接收端根据目前可用接收缓存的大小动态改变 W_T，在 TCP 流量控制中就使用这种方式。

3．自动请求重传（ARQ）

（1）停等 ARQ

1）停等 **ARQ** 的工作机制。

停等 ARQ 的基本思想是在发送端发出一个数据帧后停下来不再发送，等待接收端的 ACK 到达，ACK 到达后才发送下一帧。

停等 ARQ 实际上也使用了滑动窗口技术，它的发送窗口大小是 1，接收窗口大小也是 1，因此，在发送出去一个数据帧后，停止发送，等待接收端的 ACK。

停等 ARQ 要处理传输中可能出现的以下 3 种传输差错。

① 接收端收到了发来的数据帧，但检测出收到的帧有差错。

② 发送端发出正确的数据帧后，但传输中数据丢失，发送端不可能收到 ACK。

③ 接收端收到正确的数据帧，但发出的 ACK 丢失，发送端也不可能收到 ACK。

对于差错①，接收端丢弃此帧，并可以考虑采用下面两种方式进行重传。

① 负确认重传。但如果 NAK 丢失，发送端将收不到 NAK，又有新的问题产生。

② 超时负载。

对于②和③这两种差错，即使发送端一直等下去，也不会等到接收方的 ACK，这样就出现死锁。要解决死锁，可采用上述的方法②。重传定时时间 T_{OUT} 应大于一个帧的正常往返传输时间，主要包括数据帧的发送时间 T_{DATA}、确认帧的发送时间 T_{ACK}、链路的往返传播时延 2τ 及必要的处理时间 T_{PRO}（差错检验等）。

但对于差错③，超时重传会使接收端收到两个同样的数据帧，且接收端无法识别后者是一个数据相同的新帧还是重新的旧帧。解决重复帧的方法是为数据帧和确认帧编上序号。对于停等 ARQ，用 0 和 1 交替地编号就可以区分上述两种情况，以辨别出重复的数据帧而将其丢弃。接收端正确地收到 1/0 号数据帧，发回确认 ACK1/ACK0，确认序号表明期望接收的下一个序号。

停等 ARQ 采用超时重传的方式，并对 ACK 编号。对于传输差错①，接收端将不发送 ACK。这样，以上 3 种传输差错问题都可以解决。

2）停等 ARQ 示例。

图 2-10 所示为一个停等 ARQ 的传输过程示例。其包括了正常发送和确认、发送数据帧丢失和 ACK 丢失等情况。图中的水平方向表示了发送站 A 和接收站 B 之间的距离，垂直方向为时间，向下是时间增长方向，表示数据帧和确认帧传播的带箭头水平线都向下倾斜，反映了传播时延 τ。实际上，发送的数据帧和确认帧在垂直方向上都应该有一定的宽度，在图右侧的局部放大图中可清楚地看到，其大小反映了它们的发送时间 T_{DATA} 和 T_{ACK}，发送时间和帧长度成正比，与发送的比特率成反比。

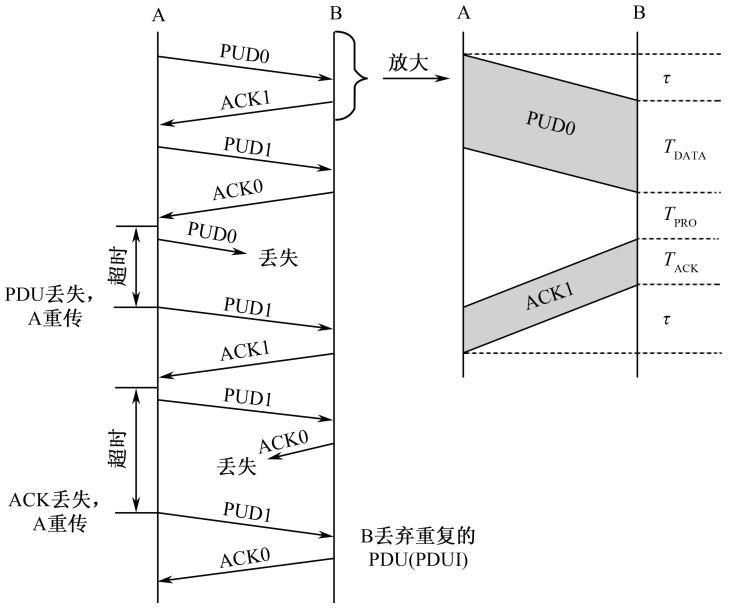


图 2-10　停等 ARQ 的传输过程示例

3）停等 ARQ 的链路利用率。

停等 ARQ 机制的问题是可能产生严重的低效率。下面对停等 ARQ 性能进行粗略的定量分析。正常传输的情况下，停等 ARQ 把传输流量控制在每一个往返时间中一个数据帧的水平，一个往返时间为 $T_{\mathrm{DATA}}+T_{\mathrm{ACK}}+2\tau+T_{\mathrm{PRO}}$。因此，链路利用率可以表示为

$$\begin{aligned}\eta &= T_{\mathrm{DATA}}/(T_{\mathrm{DATA}}+T_{\mathrm{ACK}}+2\tau+T_{\mathrm{PRO}})\\ &\approx T_{\mathrm{DATA}}/(T_{\mathrm{DATA}}+2\tau)\\ &\approx \text{帧的比特长度}/(\text{帧的比特长度}+2\times\text{链路的比特长度})\end{aligned}\qquad (2\text{-}1)$$

式（2-1）中的第 2 行是一个近似值，因为一般情况下 T_{ACK} 和 T_{PRO} 较小。将其分子和分母同乘以带宽，则变为第 3 行。由式（2-1）可见，如果帧的长度很大而传播时延又小，停等 ARQ 可以有很高的 η；如果帧的长度较小而链路的传播延时又很长（如卫星链路），η就变得很低。为此，回退-N ARQ 和选择重传 ARQ 对停等 ARQ 做了改进。

（2）回退–N ARQ

1）回退–N ARQ 的工作机制。

回退–N ARQ 是对停等 ARQ 的改进。回退–N ARQ 也使用滑动窗口机制，但 $W_{\mathrm{T}}>1$，发送

方在每收到一个 ACK 之前不必等待，可以连续地发送窗口内的帧，如果这时收到接收端发回的 ACK，还可以继续发送后续的帧，因此这种方式也称为连续 ARQ。与停等 ARQ 相比，连续 ARQ 减少了等待时间，提高了传输的吞吐量和传输效率。

回退-N ARQ 也使用超时重传机制。对于发送的每一帧设置重传定时器，发送端发出一个帧之后启动该定时器。若因发送帧丢失、出现传输差错或 ACK 丢失使定时器超时仍未收到 ACK，则要重传此帧，而且还必须重传此帧后面所有的已发帧（不管这些帧是否已传输），这正是这种机制称为回退-N ARQ 的原因。

当每收到一次失序（Out-of-order）的数据帧时，接收端都应重传上次已发送的 ACK，这可弥补上次已发送的 ACK 可能发生的丢失。

对于接收的有差错或失序的帧，回退-N ARQ 还可以使用 NAK，指明期望发送端重传帧的序号，而不必等到重传定时器到时，这样可以减少回退重传的帧数，提高传输的效率。如果 NAK 丢失，重传定时器启动重传。

回退-N ARQ 中，接收端的接收窗口中 W_R=1，也就是说，接收端不保存失序的帧，前面的帧丢失时，发送端还要重传这些失序的帧。一般地，当接收端收到一个有序且无差错的帧后，接收窗口向前滑动，准备接收下一帧，并向发送端发出一个 ACK。为了提高效率，接收端也可以使用累计确认的方式。

2）回退-N ARQ 示例。

下面结合图 2-9 及图 2-11 说明回退-N ARQ 的工作工程。图 2-9 中，W_T=5，在接收对方 ACK 之前，发送方最多可以连续发送出 5 个帧。图 2-9 中（a）、（b）和（c）3 个图的状况已经说明。图 2-11 的（a）、(b）和（c）3 个图分别与图 2-9 的（a）、(b）和（c）的发送窗口的状态相对应，表示对应的接收窗口的状态。

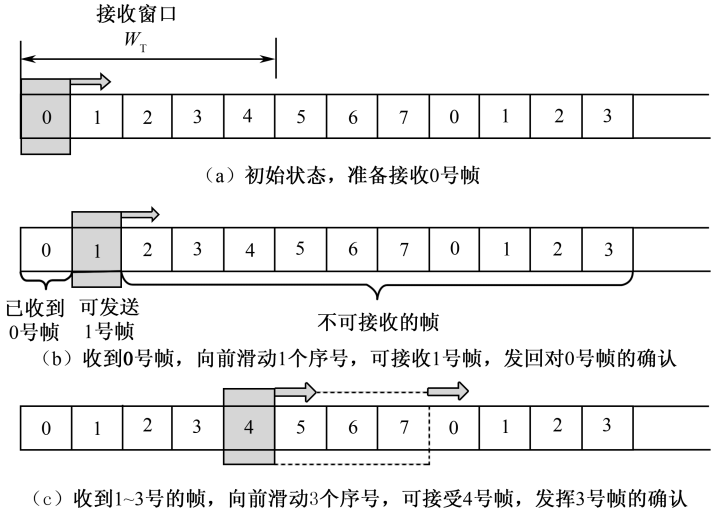

接收窗口
W_T
0 1 2 3 4 5 6 7 0 1 2 3
（a）初始状态，准备接收0号帧
0 1 2 3 4 5 6 7 0 1 2 3
已收到 0号帧
可发送 1号帧
不可接收的帧
（b）收到0号帧，向前滑动1个序号，可接收1号帧，发回对0号帧的确认
0 1 2 3 4 5 6 7 0 1 2 3
（c）收到1~3号的帧，向前滑动3个序号，可接受4号帧，发挥3号帧的确认

图 2-11　回退-N ARQ 的接收窗口

① 初始状态，发送窗口如图 2-9（a）所示；接收窗口如图 2-11（a）所示，位于 0 号的位置，准备接收 0 号帧。之后，发送端发出窗口内的 5 个帧。

② 接收端收到 0 号帧后，接收窗口向前移动一个序号，如图 2-11（b）所示，准备接收 1 号帧，同时向发送端发出对 0 号帧的确认 ACK1，使得发送端的发送窗口向前滑动了 1 个序号，如图 2-11（b）所示。

③ 接收端接收到了 1 号帧，接收窗口滑动到 2 号帧的位置，又收到 2 号帧，窗口滑动到 3 号帧的位置，又收到 3 号帧，接收窗口滑动到 4 号帧的位置，如图 2-11（c）所示。此时向发送端发出了对 3 号帧的累积确认 ACK4，使得发送端的发送窗口向前滑动了 3 个序号，如图 2-9（c）所示。

图 2-9 和图 2-11 所示的过程是正常发送和接收的情形。如果发送端发送的数据帧丢失、出错或接收端 ACK 帧丢失等，都会引发重传的过程。我们接着以上述例子继续说明。

④ 假设发送端目前进行到如图 2-9（c）所示的状态，发送窗口内有 4～7 号和 0 号帧，而且发出了 4～7 号帧，但可惜的是其中的 4 号帧丢失了。接收端的接收窗口将不再向前滑动，停留在图 2-11（c）的状态，后面接收的失序的 5～7 号帧将被丢弃，且当接收端每接收到一次失序的数据帧（5～7 号帧），都重复发送一次已发送过的 ACK4。

⑤ 因为 4 号帧丢失，接收端不可能发回 ACK5，使得发送端 4 号帧的重传定时器到时，于是要重传 4 号帧。此时发送端不仅要重传丢失的 4 号帧，而且还要重传其后已发出的 5～7 号帧。

图 2-9 及图 2-11 表示了回退-N ARQ 滑动窗口的工作过程，发、收双方各种帧的传输过程如图 2-12 所示，图 2-12 中所示的是一个双工链路。

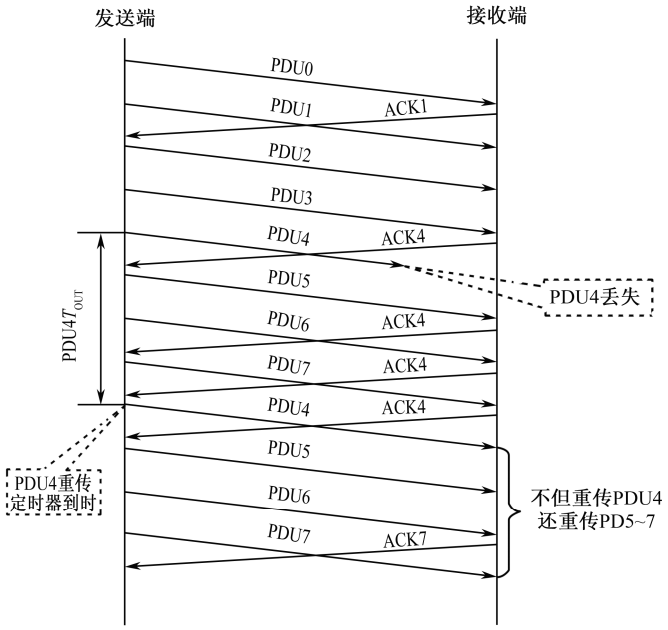


图 2-12　发、收双方各种帧的传输过程

3）回退-N ARQ 发送窗口的限制。

回退-N ARQ 对发送窗口的大小是有限制的，如果帧的序号用 n 比特编号，则发送窗口 W_T 应满足

$$W_T \leqslant 2^n - 1 = \text{最大序号} \tag{2-2}$$

例如，若 n=3，最大序号为 7，则要求 $W_T \leqslant 7$。

当 $W_T \geqslant 2^n$ 时，回退-N ARQ 将会出现某些不确定性。

4）回退-N ARQ 的链路利用率。

因为可以连续发送窗口内的多个数据帧，回退-N ARQ 提高了链路利用率。但另一方面，如果在已发送的数据帧中，有一个前面的数据帧出错，那么其后正确传送的数据帧也必须重传，浪费信道资源，这时又降低了链路利用率。可见，当信道的传输质量好、误码率很小时，回退-N ARQ 协议链路利用率大。因此，回退-N ARQ 不一定优于停等 ARQ。

为了提高链路利用率，对于比特长度大的链路，连续 ARQ 应该相应地使用大的 W_T，而且 W_T 应该使连续发送的比特长度大于链路的往返比特长度，使得传输链路处于忙碌状态。

（3）选择重传 ARQ

1）选择重传 ARQ 的工作机制。

选择重传 ARQ 也是一种连续 ARQ，在回退-N ARQ 机制的基础上做了如下两点改进。

① 接收窗口 W_R >1，这样可以接收和保存正确到达的失序帧。

② 出现差错时只重传错的帧，后续正确到达的帧不再重传，提高了信道的利用率。

在图 2-11（c）中虚线所示的选择重传 ARQ 的例子中，W_R=4，4～7 共 4 个序号落入接收窗口。即使 4 号帧丢失，后续接收的 5～7 号帧若无检验差错也要保存，而且每收到一次失序的帧，都重复一次 ACK4。待发送端 4 号帧的重传定时器到时，重传 4 号帧。接收端收到了正确的 4 号帧后，发出对 7 号帧的累积确认 ACK0，接收窗口也同时向前滑动 4 个序号。可见，选择重传 ARQ 需要接收端设置一定容量的缓冲空间。

选择重传 ARQ 也可以使用 NAK，当接收帧有错误或者失序时，指明期望发送端重传帧的序号。

2）选择重传 ARQ 发送窗口的限制。

选择重传 ARQ 中，接收窗口不应该大于发送窗口，一般应相等，即 $W_T = W_R$。而且用 n bit 对帧编号时，应该满足

$$W_T = W_R \leqslant 2^n / 2 = (\text{最大序号} + 1) / 2 \tag{2-3}$$

例如，n=3 时，最大可选 $W_T = W_R = 4$。

当 $W_T \geqslant 2^n$ 时，选择重传 ARQ 也会出现某些接收不确定性。

2.1.4 数据链路控制协议举例

本节介绍 ISO 模型中数据链路层协议，但在通信业有时也称为链路通信规程。实际上，在数据链路层中，“规程”和“协议”两词可通用。

数据链路控制协议可分为异步协议和同步协议两类。在逐位传送的串行通信中，接收端必须能识别每一个二进制位从什么时候开始，这就是位定位。通信中一般以若干位表示一个字符，除了位定时器外，还需要在接收端能识别每个字符从哪位开始，这是字符定时。异步协议把每个字符看作一个独立的信息，在每个字符起始处同步，但各个字符之间的间隔时间是可以变化

的。由于发送器和接收器近似于同一频率的两个时钟（两个时钟频率严格完全相同是不可能的），能够在一段短时间内保持同步，就可以用字符起始处同步的时钟来取样该字符中的各位，而不需要每位都严格地同步。同步协议则把许多字符组成一个数据块（前面所述的帧），除在该数据块的起始同步外，还要在后面维持固定的时钟，实际上是发送端通过某种技术将时钟混合到数据中一起发送出去，而接收端又从输入数据中分离出时钟来。该时钟不但用来定时字符内的各位，也用来定时字符本身。由上面所述可见，同步和异步主要区别在字符之间，同步是指连续传送的字符之间是同步的，而异步则指字符之间不是同步的。同步协议由于接收器能从输入数据中分离出时钟，实现起来较复杂，这个功能通常是由调制解调器来完成的。但是，下面就会看到，由于同步协议是许多字符组成一个数据帧发送的，它能有效地利用信道，也便于实现其他更强的控制功能。

1．起止式异步规程

典型的异步数据链路层协议是起止式异步规程，最早用在印字电报系统的电传机上。它是一个字符一个字符传输的，字符和字符间没有固定的时间间隔要求。因为电传机上每按一个键就产生一个字符，而字符间的间隔时间，即两次按键间的间隔时间是不定的。在传输时，每个字符之前都要有一位起始位，而后是该字符的 5～8 位数据位，这要根据采用何种字符编码而定，随后可以有 1 位奇偶校验位，也可以没有 1 位奇偶校验位。接着是 1 或两位的终止位，它是靠起始位和终止位来进行字符同步的，故称为起止式规程。

起始位取低电平（逻辑值 0），接收端是靠检测由空闲或前一字符终止位的高电平（逻辑 1）到该起始位的低电平的下降沿得知一个字符开始的。这里要指出一点，起止式规程中规定数据位中紧接着起始位的应是最低有效位，在奇偶校验位或终止位前面的是数据的最高有效位。因此，采用起止式规程传输 ASCII 字符 E（1000101），并带偶校验时，最后的终止位一定取高电平（逻辑值 1），它后面可以跟随机空闲（也是高电平，逻辑值 1）作为字符间的不定间隔，也可以直接跟随下个字符的起始位。由于空闲或终止位都规定为高电平，这就保证了起始位开始处一定有个下降沿。终止位长度可以取 1 位、1.5 位或两位。一般来说 5 单元字符终止位取 1 或 1.5 位，其他单位的字符码终止位取 1 或两位。

如前所述，接收端是通过检测起始位的下降沿来决定一个字符的开始的，并以此前沿作为取样后面各位的定时基准。大部分接收端使用比位周期更短周期的时钟来控制取样时间，假如有一个频率为位频率 16 倍的接收端时钟，利用这个时钟，接收端能在一个位周期的 1/16 时间内决定字符的开始，并按如下步骤进行后面各位的取样。

1）当检测到起始位下降沿时将某个计时器清零。

2）16 倍频时钟的每个脉冲使计数器加 1。

3）当计数器第一次到达 8 时，表示已到起始位的中间，此时取样值应为 0，并把计数器清零。若取样值不为 0，那么一开始检测到的下降沿不是真正起始位的前沿，而可能只是一次干扰，此次检测应作废。

4）以后，计数器每次到达 16 时，就取样输入波形，将取样到的数值暂存起来，并将计数器清零，取样重复直至最后的终止位被取样。

5）如果终止位取样正确（为 1），那么字符被接收，由暂存处装入寄存器。若应该为终止位处取样到逻辑值 0，说明同步或者传输有问题，该次取样的字符作废，不被接收。

虽然这个例子的接收端时钟是位频 16 倍，但接收端时钟也可以是位频的其他倍数。更高的接收端时钟频率可以得到更高的分辨率，可以允许接收端的位周期和发送端位周期间有一定的误差，只要在 10 或 11 位的范围内，误差的积累不使得对某一位的取样跳跃到另一位的区间内，上述规程就能正常工作。

为了达到这个对两端时钟差别不敏感的性能付出的代价，每个字符前面需要各加上至少一个极性相反的起始位和终止位，这样使得信道有效利用率损失了只有约 8/11，即约 80%。起止式协议一般用在机电中断设备（包括 CRT 键盘输入）、数据速率较慢的场合。

2．面向字符的同步控制协议 BSC

最早出现的是面向字符的同步规程，其典型代表是 IBM 公司的二进同步通信 BISYNC 或 BSC（Binary Synchronous Communication）协议。这个规程不像起止式规程那样，一个字符一个字符地传输，而是若干个字符组成一个信息块——帧一起发送。利用一些特殊定义的字符来界定一帧的开头与结束、分隔不同的段和控制整个信息交换过程，被传输的数据也由字符组成，因而被称为面向字符的规程。随后，为了使各制造商的产品尽可能兼容，以方便用户，美国国家标准协会和 ISO 都提出了类似的标准。国际标准化组织的标准称为数据通信系统的基本模式控制规程（Basic Mode Control Procedure for Data Communication System），即 ISO 1745。

这种规程的一般帧格式如图 2-13 所示。

	SYN	SYN	SOH	标题	STX	数据段	ETB ETX	块校 验码	

图 2-13　面向字符同步规程的帧格式

图 2-13 中的 SYN 是同步字符，每帧开始处有若干个 SYN。接收端不断检验收到的每一位，一旦发现出现同步字符后，就知道是一帧的开始。接着的 SOH 标志着标题的开始，也是一种特殊定义的字符，称为序始字符。标题中可以包括源地址、目标地址和路由指示等有关的信息。STX（Start of Text）称为文始字符，该特殊定义的字符标志着传送数据（正文）的开始。数据段中就是要传送的正文本身的内容，数据段后面可以是组终字符（End of Transmission Block，ETB）或文终字符。ETB 用在正文很长、需要组成若干个数据段、在不同帧中发送的场合，除去最后一个数据段用 ETX 外，其余数据段都要用 ETB。块校验码可以通过横向奇偶检验，块校验码也可以是 CRC 码，美国采用的生成多项式为 CRC-16 和 CRC-12，其他国家大多采用 CRC-CCITT，校验的范围从 SOH 直到 ETX 或 ETB。

特殊定义的字符 SYN、SOH、STX、ETB 和 ETX 等具体的二进制位模式随采用的字符编码集不同而异。面向字符的同步协议中用到的特殊字符除上述 5 个外，还有 EOT、ENQ、ACK、NAK 和 DLE。它们的名称及在 ASCII 和 EBCDIC 两种字符编码集的二进制位模式列在表 2-1 中，限于篇幅，除了 DLE 字符外，不再详细介绍其他特殊字符的用法，它们都是用来在通信双方间传送一些控制信息，统称为通信控制字符。

表 2-1　通信控制字符

名　　称	ASCII	EBCDIC
序始字符（SOH）	0000001	00000001
文始字符（STX）	0000010	00000010
组终字符（ETB）	0000111	00100110
文终字符（ETX）	0000011	00000011
同步字符（SYN）	0010110	00110010
送毕字符（EOT）	0000100	00110111
询问字符（ENQ）	0000101	00101101
确认字符（ACK）	0000110	00101110
否认字符（NAK）	0010101	00111101
转义字符（DLE）	0010000	00010000

同步规程中为了避免异步方式中的许多起始和终止信号，信道的有效利用率得到了提高（特别是在传输较长的数据时）。同时，由于有标题字段，可以具备更为复杂的通信控制能力，校验的功能也得到增强。在同步方式中，接收端要在相当长的数据流中保持位定时，因而其时钟必须和发送端时钟保持一致，通常是通过调制解调器将时钟混合到数据流中，接收端又通过调制解调器从数据流中提取出时钟信号。面向字符的同步规程的最大缺点是它和特定的字符编码集的关系过于密切，不利于兼容。为了实现“数据透明”而采用的上面所述的字符填充方法，无论用硬件还是软件，实现起来都比较麻烦，而且它也要依赖于所采用的字符编码集。正是由于它的上述缺点，随着数据通信和计算机网络的发展，20 世纪 70 年代又产生了新的面向比特的同步规程。

3．面向比特的同步控制协议 HDLC

20 世纪 70 年代初，IBM 公司率先提出了面向比特的同步数据链路控制规程 SDLC（Synchronous Data Link Control），并将它们交给美国国家标准协会和国际标准化组织。国际标准化组织和美国国家标准协会将其发展，分别成为各自的标准：高级数据控制规程 HDLC 和先进数据通信控制（Advanced Data Communition Control Procedure，ADCCP）规程。与此同时，X.25 建议中的第二层也采用了 HDLC 的一种变体，称为 LAP（Link Access Procedure）或 LAPB（Link Access Procedure Balanced）。这些不同的规程间尽管存在着许多细微的差别，但总的来说是大同小异的。这些规程的帧中所传输的数据可以含有任意数量的位，不必是某一特定字符码的整数倍，而且它是靠约定的比特模式，而不是靠使用特定定义的字符（字符的编码与字符密切相关）来定界开始和结束的，故称为“面向比特”的，这种规程的一般帧格式如图 2-14 所示。

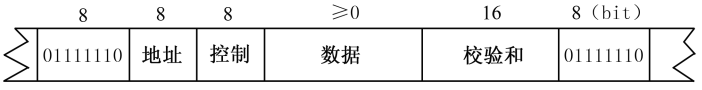


图 2-14　面向比特的同步规程的帧

面向比特规程中的“数据透明”是通过比特填充来实现的。发送器对除了标志以外的其余字段，每连续出现 5 个“1”之后，就自动插入 1 个“0”。这样一来，即使数据字段中出现了如标志“01111110”那样的比特模式，通过“0”插入后，线路中出现的是 011111010，接收器就能把它与标志区分开来。当然，接收器在接收数据的过程中还要执行 1 个“0”删除的相反过程，也就是在连续出现 5 个“1”后，要删除 1 个“0”。图 2-15 给出了一个比特填充的例子，比特填充通过硬件电路不难实现。

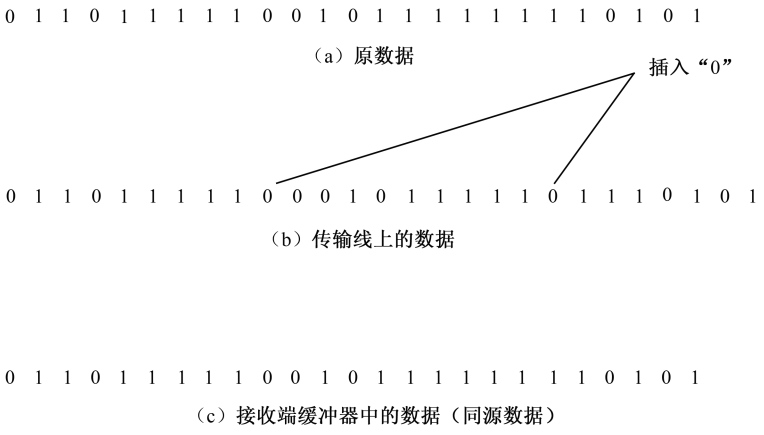


图 2-15　比特填充

HDLC 是一种典型的面向比特的同步规程，其帧的一般格式已在图 2-14 中给出。可见，接收端可以通过搜索一个特定的比特模式“01111110”来定位帧的开头和结束，这个特定的比特模式称为标志。帧中紧接着的地址字段一般为 8 位，在某些规程中还允许扩充为 16 位，甚至更多。在多点连接的通信中，地址是必不可少的，例如，一台计算机通过一条线路连接到多个终端时，该地址用来指明此次是和哪一个终端通信。在点到点连接线路中，这个地址字段可用来区分通信的发起方和响应方。例如，在 X.25 的 LAP 和 LAPB 中规定：11000000（称为地址 A）表示 DCE 向 DTE 发送的命令或 DTE 向 DCE 发送的响应；10000000（称为地址 B）表示 DTE 向 DCE 发送的命令或 DCE 向 DTE 发送的响应。

地址字段后面的控制字段一般为 8 位，某些情况下可扩充为 16 位。紧接着是任意比特长度的数据。在帧尾标志前面的是 16 位的循环冗余检验码，对从地址开始直至数据字段的内容进行校验。

控制字段的第 1 位或第 1、2 位用来区分 3 种不同类型的帧：信息帧（I-帧）、监控帧（S-帧）和无编号帧（U-帧），如图 2-16 所示。

控制字段的第 5 位都是 P/F 位，即探询/结束（Poll/Final）位。当主站发出一帧命令时，该位起探询的作用，该位为“1”时，将要求被选择的从站给出响应，从站的响应可以是多少帧，最后一帧的位为“1”，指示“结束”。

信息帧用来传送用户的数据。面向比特的规程为了使得发送端不必在发送一帧后停下来，等待接收端确认信号回来后再发送第二帧，都允许连续发送若干帧。这些帧的编号存放在 N(S)

中，它有 3 位，值取 0～7，这样做可以避免停顿等待而浪费信道。在双向通信中，接收端对某一编号帧已收到的确认信息可通过反向送的信息帧捎带回去，它存放在 N(R)中。例如，N(R)=5，就表示下一帧要接收 5 号帧，换句话说，确认 5 号帧之前的 4 号帧、3 号帧、2 号帧都已正确地收到。在远距离传输和信道速率很高时，允许发送端连续发送更多尚未收到确认的帧，可以将控制字段扩充为 16 位，此时 N(S)和 N(R)都可分别取 7 位。

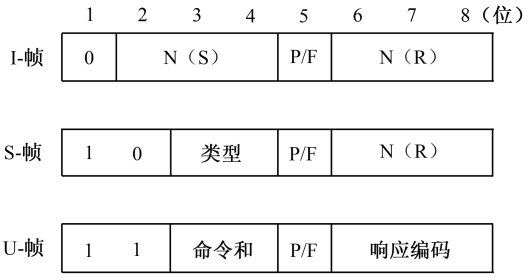


图 2-16　控制字段的格式

监控帧用来管理和控制链路操作。通常 S-帧不带数据，只有 6 字节，即 48 位。监控类型共两位，有 4 种不同的编码，它们分别表示如下。

00——接收准备好（RR），例如，控制字段为 10001011，由主站发出的该 S-帧可表示主站已收到从站发来的编号为 2 的帧，并希望从站发来编号为 3 的帧。

01——拒绝（REJ），它要求发送端对从编号 N(R)开始的 I-帧重新发送。

10——接收未就绪（RNR），表示编号小于 N(R)的帧已被收到，但目前正处于忙碌状态，尚未准备好接收编号为 N(R)的 I-帧，这可用来对通信流量进行控制。

11——选择拒绝（SREJ），它要求发送端重发编号为 N(R)的单个 I-帧。

无编号帧因其帧控制字段中不含编号 N(S)或 N(R)而得名，它们用来提供各种附加的链路控制命令和响应的功能。这些不同的命令和响应可由 5 位来编码，共 32 种，但有许多是未定义的。例如，若该 5 位（图 2-16 中 U-帧的第 3、4、6、7 和 8 位）为 11001，则表示复位（RESET）命令，即要求对方各种计数器，包括 N(S)和 N(R)的值都恢复为初始值。又如，00000 表示无编号信息(UI)，这种 U-帧可带数据字段，用来传递信息，但这些信息是不提交给高层用户的，即不是用户数据。其他各种 U-帧的命令或响应含义在此不再详述。

由前面介绍可见，面向比特的规程不依赖字符编码集；比特填充的方法由硬件实现较方便；它可以用于双向同时通信；不必停等确认而可连续发送，因而可达到高数据速率；还能实现各种较完善的控制功能。所以，面向比特的规程目前得到了广泛应用。

4．BSC 和 HDLC 特点的比较

（1）适用场合

就系统结构而言，HDLC 适用于点到点或点到多点式的结构，BSC 同样也能适用于这些结构；就工作方式而方，HDLC 适用于半双工或全双工，而 BSC 则更适用于半双工方式（也可扩充为全双工）；就传输方式而言，BSC 和 HDLC 两者都只用于同步传输。在传输速率方面，BSC 和 HDLC 虽然都没有限制，但由于它们各自的特点所定，通常 BSC 用于低、中速传输，而 HDLC 则常用于中、高速传输。

（2）传输效率

HDLC 开始发送一帧后，就要连续不断地发完该帧，而 BSC 的同一数据块中的不同字符之

间可能有时间间隔，这些间隔用 SYN 字符填充。HDLC 可以同时确认几个帧，而 BSC 则在发完一数据块后必须要等待确认（“停—等”方式）。HDLC 中的每个帧都含有地址字段 A，在多点结构中，每个从站只接收含有本站地址的帧，因此，主站在选中一个从站并与之通信的同时，不用拆链，便可选择其它的站通信，即同时与多个站建立链路。而在 BSC 中，从建链开始，两站之间的链路通道就一直保持到传输结束为止。由于以上特点，HDLC 的传输效率高于 BSC 的传输效率。

（3）传输可靠性

HDLC 中所有的帧（包括响应帧）都有 FCS，在 BSC 的监控报文中只有字符校验能力而无块校验能力。HDLC 中的 I–帧按窗口序号顺序编号，BSC 的数据块不编号。由于以上特点，HDLD 的传输可靠性比 BSC 高。

（4）数据透明性

HDLC 采用“0 比特插入法”对数据实现透明传输，传输信息的比特组合模式无任何限制。BSC 用 DLE 字符填充法来实现透明传输，依赖于采用的字符编码集，且处理复杂。

（5）信息传输格式

HDLC 采用统一的帧格式来实现数据、命令、响应的传输，实施起来方便。而 BSC 的格式不统一，数据传送、正反向监控各规定了一套格式，给实施带来了不便。

（6）链路控制

HDLC 利用改变一帧中的控制字段的编码模式来完成各种规定的链路操作功能，提供的是面向比特的传输功能。BSC 则是通过改变控制字符来完成链路操作功能，提供的是面向字符的传输功能。

2.2 网络层

网络层是 OSI 模型中的第三层，介于传输层和数据链路层之间，它在数据链路层提供的两个相邻端点之间的数据帧的传送功能上，进一步管理网络中的数据通信，将数据设法从源端经过若干个中间节点传送到目的端，从而向传输层提供最基本的端到端的数据传送服务。

2.2.1 网际协议

IP 是网络层的主要协议。它的主要功能有无连接数据报传送、数据报路由选择和差错控制。IP 将报文传送到目的主机后，不管传送正确与否都不进行检验，不进行确认，也不保证分组的正确顺序，这些功能由 TCP 完成。

1．IP 数据报的格式

在 IP 中传输的数据单元称为 IP 数据报，IP 的数据报由报头和数据两部分组成。报头由一个 20 字节的固定长度部分和一个可选任意长度部分组成。数据报是理解 IP 的基础，数据报概

念类似于物理网络中的“帧”，但又与帧不同。“帧”由硬件识别，而数据报由软件识别，数据报从一台机器传送到另一台机器还是以物理“帧”进行传送。IP 支持的最长数据报为 65535 字节，具体大小可以通过软件来设置。若数据报过大，不适宜物理网络的传送，IP 会将它分解成几个较短的数据报，在传输的另一端，再把分解过的数据报按顺序重装起来。将数据包装成符合物理网络要求的帧的格式的过程称为封装。在理想情况下，完整的数据报能全部插入到一帧中，以提高传输效率，但若数据报过大，就要将其分为若干片或段了。不同物理网络对帧大小的限制是不同的，这个限制称为物理网络的 MTU，如以太网限制传送帧中的数据为 1500 字节，基于 X.25 的广域网把 MTU 限制在 128 字节，有的物理网络的 MTU 可能会更小。IP 数据报的格式如图 2-17 所示。

图 2-17　IP 数据报的格式

（1）版本字段

版本字段指示每个 IP 数据报的 IP 版本，占 4bit，若版本号不同：则需进行转换。

（2）报头长度字段

报头长度字段占 4bit，指示报头的长度。以 32bit 为一个基本单位，它的最小值是 5，也就是说，报头长度最小为 5 个 32 位长（20 字节，不包括选项和填充字段）。这个 4 位字段的最大值为 15，它限制了报头最大长度为 60 字节，因此选项最大为 40 字节。对于某些选项，如记录分组所经过的路由，40 字节往往不够，在这种情况下，这些选项也变得毫无用处。

（3）服务类型字段

服务类型字段占 8bit，指示如何处理数据报。例如，指示数据报的优先级及所需的服务类型。该字段前 3 位（从左到右）指示优先关系，共可设置 8 级，“7”表示最高优先级，“0”表示最低优先级，接下来的 4～6 位分别指示低延时、高吞吐量和高可靠性。取值为 1 时，数据报所期望的服务类型有效。若是数字化语音传输，更对速度有准确性要求，余下的两位未用。

（4）数据报长度子段

数据报长度子段占 16 位，指示了整个 IP 数据报的长度，包括报头和数据，最大值取 65535，即最大 IP 数据报长度为 65535 字节。IP 数据报长度的确定，受物理网络的限制，对大型数据报，IP 层还需要将其分片传输，每片中都由报头和数据片组成。

（5）标识字段

标识字段占 16 位，用来控制分片重装，每个数据报不管分成多少片，都具有相同的标识号，用来确定该分片属于哪个数据报，分片到达时，目标主机根据主机标识号、片偏移和源地址进行重装，每个数据报都有唯一的标识。分片是为了满足物理网络层所能传输的帧长度的上限。紧跟着标识字段的两位未用。然后是两个 1 位子段。DF 代表该数据报不能被分片，因为目标主机（或路由器）不能重装分片。MF 取“1”时代表了该分片之后，还有分片；取“0”时标识最后一个分片。

片偏移占 13 位，表示分片在当前数据报中的位置。以 8 字节为分片的基本单位，该字段的取值范围为 0～8191，即数据报最大可分为 8192 个分片（65536/8）。由于数据分片不能保证按序到达，故目标主机（路由器）用标识号和片偏移重装数据报。

（6）生存时间字段

生存时间字段占 8 位，用来确定数据报被允许在网络系统中传输最多可用多少秒。其作用是避免因网络中出现循环环路路由而使数据报长期存在（即防止数据报在网络中无限制循环），其最大值为 255s。随着数据报的传输，此值不断减小，以秒为单位。当减少到 0 时，将丢弃此数据报。

（7）协议字段

协议字段占 8 位（1 字节），用来指示传输协议类型，如“6”代表 TCP，“7”代表 UDP 等。协议的编号在整个 Internet 上是全球通用的，它定义于 RFC 1700 中。

（8）报头检验和字段

报头检验和字段占 16 位，用来确保数据报头的无差错传输，但它并不对整个数据进行校验。

（9）源 IP 地址和目标 IP 地址

它们指明了网络号和主机号，网络上的每个收发数据报的设备都必须拥有独一无二的 IP 地址。

（10）选项字段

选项字段主要用于网络控制（测试和调试）。因为报头为 4 字节的整数倍，所以当选项字段用不完 40 字节时，将用“0”填充。设置选项域的目的是补充原来 IP 的不足和后续协议的需要，允许对一些新的概念进行实验，并可避免对实际需要的信息再分配报头位。目前已定义了 5 种可选项，如表 2-2 所示。

表 2-2　5 种可选项

选项类型	描　　述
安全性	指明数据报的机密程度
严格路由选择	给出完整的路由
松散路由选择	给出一个不能漏掉的源路由表
记录路由	使每个路由器都附加上它的 IP 地址
时间标记	使每个路由器都附加上它的 IP 地址和时间标记

- 安全性用来指示数据报的机密程度。实际上，所有路由器都忽略了此字段。
- 严格路由选择给出了一个从源到目标的 IP 地址序列，要求数据报严格沿指定的路由表传输，其主要功能是路由选择表崩溃时，可由网络管理软件发出一个紧急包或做定时检测。

- 松散路由只是指定一个必须经过的路由器（或路由表），其间可经过其他路由器，即相当于严格路由中的几个关键点，关键点之间可以经过其他路由器。
- 记录路由是让沿途经过的路由器都将其 IP 地址填入选择项字段，主要用于分析路由器算法。当 ARPR 网建立之初，没有数据报经过 9 个以上路由器的情况，所以 40 字节的选项是合理的（每个 IP 地址为 32 位，9 个 IP 地址则需 36 字节，这也是报头长度为什么只有 4 位的原因（2^3=8<9，2^4=16>9）。
- 时间标记选项除了用于记录经过路由器的 IP 地址外，还记录数据经过的国际标准时间，单位为微秒，这一选项也主要用于检测路由算法。

2．IP 地址

（1）IP 地址的分类

TCP/IP 用 IP 地址来标识源地址和目标地址，但源和目标主机却位于某个网络中，故源和目标地址都有网络号和主机号，但这种标号只是一种逻辑编号，而不是路由器和计算机网卡的 MAC 地址。若局域网不与 Internet 相连，则可自定义其 IP 地址，但若要将局域网连到 Internet 中，则必须向网络信息中心（NIC）申请 IP 地址。Internet 上的每个主机和路由器都有一个 IP 地址，IP 地址具有全球唯一性。典型的 IP 地址为 32 位，常用带点的十进制标记法书写，在这种标记法中，每字节用十进制表示，其范围是 0～255。例如，设有 IP 地址 01010010 10101010 10101010 11101010，则记为 82.170.170.234。注意，这只是一种记法，实际上 IP 地址是一个 32 位二进制串。连接若干个网络的路由器，在各个网络上都对应一个 IP 地址。例如，若连接 10 个网络，则该路由器需要有 10 个 IP 地址，即同一设备所接的各个网络必须为它分配一个该网络的 IP 地址。

IPv4 的地址在使用初期采用有类型的地址，IP 地址的主管机构把 IP 分为 A、B、C、D 和 E 这 5 种类别。它们用 4 字节的第一字节（地址的高位）的前几个比特来区分不同类型的地址，如图 2-18 所示。例如，A 类地址的最高位为 0，B 类地址的最高位和次高位为 10。其中图 2-18（a）用二进制的格式写出了 5 类地址的第一字节的前几比特值（从 1bit 到 4bit），图 2-18（b）给出了 5 类地址的第 1 字节十进制数的范围。例如，B 类地址的第一字节取值范围用二进制表示是 10000000～10111111，用十进制表示为 128～191，其余的表示方法相同。其中 E 类地址作为保留地址，以准备以后再用。D 类地址用作多播地址，即在 IP 层上一个源的分组发给多个目标接收，是一对多的通信。这多个目标接收端构成一个组，这个组的地址称为组地址，所以多播又称组播。其余 3 类地址 A、B、C 类用作单播（一对一）通信的地址。

IP 地址是具有两层次的地址结构。A、B、C 这 3 类地址由 3 个字段组成，如图 2-19 所示，即除了几比特的地址类型标识符以外，将余下的比特分为两个字段，一个是代表主机所在的网络，称为网络标识符 netid，另一个字段是主机，称为主机标识符 hostid。A、B、C 3 类地址的区别就在于 netid 和 hostid 的长度划分不同。A 类地址用 7bit 作为网络标识符，共有 2^7=128 个网络。而主机标识符用 3 字节，表示在一个网络内可以有 2^{24}=16777216，即约有 1600 万个主机，说明使用 A 类地址能够构成的网络数很少，但是单个网络的规模非常巨大。B 类地址的类型和网络占用 2 字节，用 2 字节表示主机。用于 netid 的是 14bit，所以 B 类地址的

网络数是 2^{14}=16384，即约有 16000 个网络，主机数是 2^{16}=65536，每个网络最多约可以有 6 万多台主机，这是中等规模的网络。C 类地址只用 1 字节代表主机，用 21bit 表示网络，所以它的网络数为 2^{21}=2097152，即约 200 万个网络，每个网络内的主机数是 2^{8}=256，表明 C 类地址适用于有大量网络数的小型网络。

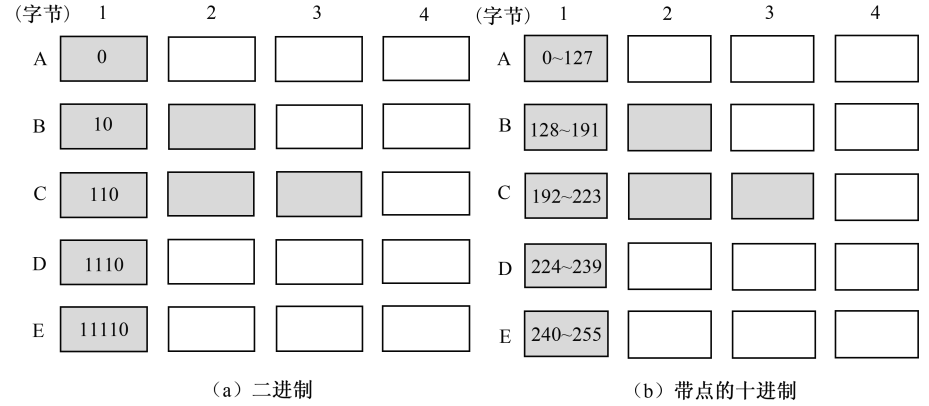


图 2-18　不同类型的地址

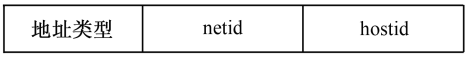


图 2-19　IP 地址结构

（2）子网

假如一个单位或一个组织获得一个 B 类地址，它只能组建一个多达 6 万台主机的网络，可是这个单位要求组建多个网络，每个网络内的主机数又大多超过 256 台，用多个 C 类地址又不行，这时该如何办？这时可以由单位或组织在它的内部采用划分子网的办法来解决。在这个单位内的网络又增加一个层次，地址变成 3 层结构。它是把 B 类地址的 16bit 主机标识符字段再划分为子网和主机号两部分。例如，某个大学有 35 个系，每个系为一个网络，则可以用 6bit 作为子网号，用余下的 10bit 作为主机号。这个大学从 Internet 方面看来虽然只是分配了一个 B 类地址，但在大学内部来讲，最多可以组建 64 个子网，每个子网在 Internet 上最多可以有 1024 台主机，图 2-20 说明了这样的地址结构，而图 2-21 说明了它在一个校园网中的采用。为了实现对子网的支持，学校的主路由器需要一个子网掩码（Subnet Mask）。子网掩码的长度也为 32bit，它代表了“网络号+子网号”与主机号的划分方案。对于上面的例子，类型+网络号+子网号共用了 22bit，主机号占 10bit。其子网掩码为 11111111 111111111 11111100 00000000，或者写出点分十进制形式，即 255.255.252.0。

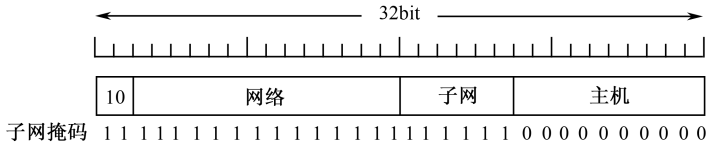


图 2-20　一个 B 类地址划分为 64 个子网

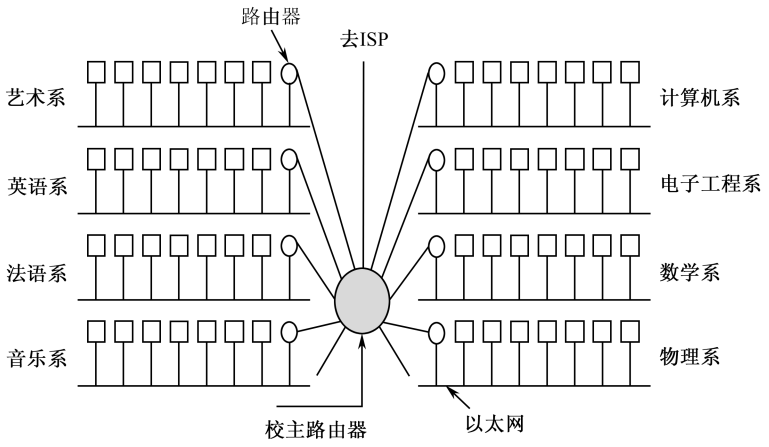


图 2-21　校园网的结构

也可以用/22 来表示上面的掩码值，它表示这个掩码是由左边的连续 22 位值为 1 的比特后随 10bit 的 0 组成，也就是掩码中连续 1 的个数是“类型+网络+子网”所占比特的长度之和，主机号的比特长度是掩码后面 0 的个数。在网络的外部，子网是不可见的。例如，主路由器从 ISP 接收一分组，其目的 IP 地址是 130.50.15.6，在外部，它是一个 B 类地址，是两层地址结构；130.50 代表了地址类型和网络号，而后 2 字节 15.6 代表了一个主机号。可是主路由器对于校园内部来讲，应解释后 2 字节为子网号+主机号。那么主路由器如何知道这个分组是送往哪个子网上的哪台主机呢？这只要将这个目的地址与子网掩码逻辑与，即可得子网号。取掩码的反码，再与目的地址相与，就可得子网中的主机号。算法如下。

130.50.15.6 AND 255.255.252.0→130.50.12.0

将 12.0 这个 2 字节的十进制数用二进制表示为 00001100 00000000，可知它的子网号为 3。

130.50.15.6 AND 0.0.3.255→0.0.3.6

将 2 字节十进制数 3.6 用二进制表示为 00000011 00000110，可知该目的主机是子网 3 的 774 号主机。

（3）广播地址和多播地址

广播（Broadcast）地址和多播（Multicast）地址不是某一台具体的机器，而是指满足一定条件的一组机器。广播地址和多播地址只能作为 IP 报文的目的地址，表示报文的一组接收者。

在每一个子网中，主机地址部分为全 0 或全 1 的地址称为直接广播地址。使用直接广播地址，主机可以向任意网络直接广播数据。很多 IP 利用这个功能向一个子网上广播地址。例如，在子网地址 166.111.64.0 的网络上，166.111.64.0 和 166.111.95.0 就是广播地址。另外一台 IP 地址 166.111.1.3 的主机，虽然不在子网上广播地址，也可以向该子网广播数据。每台主机和路由器都要接收和处理目的地址为广播地址的包。

32bit 全为 1 的 IP 地址 255.255.255.255 用于本网广播，该地址称为有限广播地址，使用有限广播地址，主机可以向本网所有的主机发送信息。

多播地址则指定一个主机组。它和广播地址的区别在于，广播地址是按主机的物理位置来划分各组的，而多播地址指定一个逻辑组，参与该组的机器可以遍布整个 Internet，多播地址可

用于电视会议等应用。

一个多播 IP 地址唯一标识一个组。每一个要求参与多播接收的主机使用 Internet 组管理协议（Internet Group Management Protocol，IGMP）将主机登记到希望加入的组中。网络中的路由器根据参与的主机位置，使该多播发送组形成一棵树。组中的成员在要发送数据时，只需发送一份数据，IP 报文目的地址为相应的多播地址。路由器根据已经形成的发送树依次转发，只是在树的分岔点复制数据，并向多个网络发送。经过多个路由器的转发后，则可以到达所登记到该组的主机，这样就大大减少了源端主机和网络的负载。

（4）IP 地址转换

1）地址解析协议。

假设在以太网上运行的 IP 把需要发送的数据封装后，要交给数据链路层发送。一般而言，以太网上使用的是 6 字节的 MAC 地址，每一个网卡上使用的 MAC 地址是由网卡的生产厂家设置的，它与该接口上的 IP 地址没有对应关系。IP 只知道要发送的下一站的主机和路由器的 IP 地址，那么数据链路层如何决定下一站主机的 MAC 地址呢？在以太网等局域网上，使用地址解析协议（Address Resolution Protocd，ARP）来实现 IP 地址到 MAC 地址的动态转换。

假设在一个以太网上，有两台计算机 A 和 B。A 和 B 之间要通过 TCP/IP 通信，则双方必须知道对方的 MAC 地址。每台主机都要维护一个 IP 地址到 MAC 的转换表，该表即为 ARP 表。其中存放着最近用到的一系列和它通信的同网计算机的 IP 地址和 MAC 地址的映射。在启动主机时，ARP 表为空。

假设源端 A 要和 IP 地址 129.1.1.1 的主机 B 通信。A 首先查看自己的 ARP 表，看其中是否有 129.1.1.1 对应的 ARP 表项。如果找到了，则不用发送 ARP 包，而直接利用 ARP 表中的 MAC 把 IP 数据包进行帧封装，发送给目的地。如果在 ARP 表中找不到对应的地址项，则把该数据报放入 ARP 发送等待队列，然后 ARP 创建一个 ARP 请求，并以广播方式发送（把以太帧的目的 MAC 地址设置为 FF-FF-FF-FF-FF-FF）。ARP 数据报的格式如图 2-22 所示。在 ARP 请求报中有要请求的计算机的 IP 地址，以及主机 A 自己的 IP 地址和 MAC 地址。因为是以广播方式发送的，所以所有网上的计算机都可以接收到，不过仅被请求的主机（即主机 B）处理。主机 B 首先把 ARP 请求包中发起者的 IP 地址和 MAC 地址放入自己的 ARP 表中，然后组织 ARP 响应包，在报中填入自己的 MAC 地址，发送给主机 A。这个响应不再以广播的形式发送，而是在以太帧的目的地址字段填入 A 的 MAC 地址，直接发送给主机 A。

图 2-22　ARP 数据报的格式

主机 A 在接收到响应后，将从报中提取出目的 IP 地址及其对应 MAC 地址，加入自己的 ARP 表中，并且把放在发送等待队列中的所有数据包都发送出去。如果一条 ARP 项很久没有使用，则将从 ARP 表中删除，这样可以节省内存空间和 ARP 表的查表时间。

需要特别指出，ARP 是解决同一个局域网上的主机或路由器的 IP 地址和硬件地址的映射问题，在同一个局域网内，数据报中的目的 IP 地址和目的链路层地址都指的是相同的主机。如果所要找的主机和源主机不在同一个局域网上，那么这时就要借助于网络层的协议，配合链路层协议才能将数据报成功地发送到目的主机上，此时每个链路层可能具有不同的数据帧首部，而且链路层的目的地址（如果有的话）始终指的是下一站（路由器）的链路层地址，这时数据报中的目的 IP 地址和目的链路层地址并不相同。那么就可能会产生这样的问题：既然在网络链路上传送的帧最终是按照硬件 MAC 地址找到目的主机的，那么为什么我们不直接使用硬件地址进行通信，而是要使用抽象的 IP 地址并调用 ARP 来寻找出相应的硬件地址呢？

这个问题必须弄清楚。由于全世界存在着各式各样的网络，它们使用不同的硬件地址。要使这些异构网络能够互相通信，就必须进行非常复杂的硬件地址转换工作，因此几乎是不可能的事，但统一的 IP 地址把这个复杂问题解决了。连接到 Internet 的主机都拥有统一的 IP 地址，它们之间的通信就像连接在同一个网络上那样简单、方便，因为调用 ARP 来寻找某个路由器或主机的硬件地址都是由计算机软件自动进行的，而用户是看不见这种调用过程的。设想有两个主机可以直接使用硬件地址进行通信（具体实现方法暂不必管），再假定其两个主机的网卡都同时坏了，然后又都更换了一块，因此它们的硬件地址也都改变了。这时，这两个主机怎样能够知道对方的硬件地址呢？显然很难。但 IP 地址独立于主机或路由器的硬件地址，硬件地址的改变不会影响使用 IP 的主机的通信。因此，在虚拟的 IP 网络上用 IP 地址进行通信给广大计算机用户带来了很大方便。

下面简要说明 ARP 包中每一个字段的含义。

- 硬件类型：表示其他物理网络的（硬件接口）类型，如 X.25、ATM、以太网、FDDI 网等。
- 协议类型：表示网络协议类型。
- 头长度：表示 ARP 报文的总长度。
- 协议地址长度：对于 IP 来说，该字段的值应该为 4。
- 操作：表示 ARP 请求和响应。
- 源端物理地址：源端的物理地址，由请求者填充。
- 源端协议地址：源端的 IP 地址，由请求者填充。
- 目的端物理地址：在请求包中应该为 0，在响应包中则被设置为目的端的物理地址，由发送响应包的主机填充。
- 目的端协议地址：由 ARP 请求者填充，其中含有所希望知道的主机的 IP 地址，只有与该地址匹配的主机才发送响应。

ARP 不是 IP 的一部分，它不包含 IP 头，而是直接放在以太网帧的数据部分。并且，在以太网中定义了一种新的以太类型来标识 ARP 包，ARP 请求包和响应包的以太类型都是 0x0806。

2）反向地址解析协议。

反向地址解析协议（Reverse Address Resolution Protocol，RARP）可以实现 MAC 地址到 IP 地址的转换。无盘工作站在启动时，只知道自己的网络接口的 MAC 地址，而不知道自己的 IP 地址。它首先要使用 RARP 得到自己的 IP 地址后，才能和其他服务器通信。

在一台无盘工作站启动时，工作站首先以广播方式发出 RARP 请求。这个 RARP 服务器就会根据提供的 RARP 请求中的 MAC 地址为该工作站分配一个 IP 地址，组织一个 RARP 响应包发送出去。RARP 包和 ARP 的格式完全一样，唯一的差别在于 RARP 请求包中是发送者填充的源端物理地址，而源端 IP 地址为空。在同一个子网上的 RARP 服务器接收到请求后，填入分配的 IP 地址，然后发送回源端。

3．IP 路由表

（1）路由表的格式

在同一个子网上的两台计算机使用 IP 通信时的过程很简单。首先利用 ARP 得到对方的 MAC 地址，然后利用 MAC 地址把要传输的 IP 数据包进行封装，交给数据链路层发送即可。如果源端和目的端在不同的子网上，则数据必须经过路由器转发。

这时，对于发送端来说，首先要确定目的端是否和它在同一个子网上。如果目的端在不同的子网上，则源端要决定到底应该向哪一台路由器递交要转发的数据。路由器得到要求转发的数据后，要为之选择路径，确定应该向哪一个下一站系统发送。

上述这些工作，都可以通过一个 IP 路由表来完成。在主机和路由器上，路由表的形式大体上是相同的。

图 2-23 说明了 IP 数据包的转发过程。

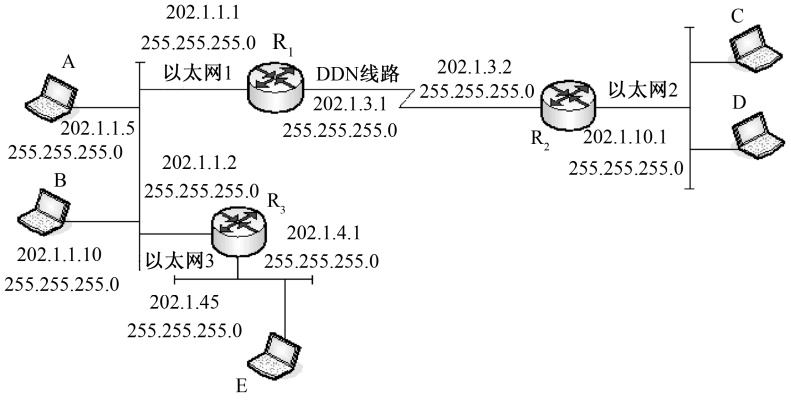


图 2-23　IP 数据包的转发过程

图 2-23 中主机 A 的路由表格式和内容如表 2-3 所示。

路由表具体的格式随操作系统的不同而有所差异，但是在每个路由表中，至少应有目的地址、掩码、网关及接口名称等。目的地址和掩码是整个表的关键字，唯一地确定到某一目的地路由。网关表示下一站路由器的地址，而接口则表示应该转发的网络接口的名称。在路由表中

每项还有两个标志值得说明，标志 H 表示该路由是主机路由，即该路由项指明到一台具体的主机的路由，G 则表示网关字段中的地址是一个有效路由器的地址。

表 2-3　主机 A 的路由表格式和内容

目的地址	网　关	掩　码	标　识	MSS	窗　口	接口名称
127.0.0.0	0.0.0.0	255.0.0.0	H	3548	0	10
202.1.1.0	0.0.0.0	255.255.255.0	H	1500	0	Eth0
202.1.4.0	202.1.1.2	255.255.255.0	HG	1500	0	Eth0
0.0.0.0	202.1.1.1	0.0.0.0	HG	1500	0	Eth0

上述路由表给出了 4 条路由信息。这 4 条路由信息可简述如下。

第 1 条是到 127.0.0.0 的路由。127.0.0.0 子网为回送子网，而 127.0.0.1 为回送接口，所以网络接口上发送数据都要交给主机去处理。如果数据包的目的地址为 127.0.0.1，则和该路由器匹配，因而 IP 把它交给虚拟的回送接口（用 10 表示）去处理。

第 2 条路由的目的子网实际上是和主机直接相连的子网地址。目的地址为该子网的子网号，而掩码为该网络接口上的掩码。因为没有设置标识位 G，所以这表示网关字段并不是一个真正的路由器地址。此时数据包应该往 Eth0 口发送，而下一跳的地址应该是 IP 数据包中的目的地址。

第 3 条路由是以太网 3 的路由。路由项中的 G 标识位有效，表示到达该网络应该经过路由器 202.1.1.2。

第 4 条路由的目的地址和掩码全为 0，表示和任何目的地址都可以匹配。这样的路由称为默认路由，它表示如果目的地址和路由项中的其他项都不匹配，则使用该项作为其路由。这条路由设置了标志位 G，所以路由项中的网关字段是有效路由器地址，此为下一跳路由器的地址，并且该路由器在 Eth0 接口上。

（2）路由表的查找

如何确定一条路由是否符合要求呢？办法很简单，即将 IP 报文中的目的地址和路由项中的掩码做与运算，看结果是否与相应的路由项中的目的子网地址相等即可。

在查找路由表时，要求使用最佳匹配原则。因为在路由表中每条路由的掩码长度不一样。如果有多条匹配的路由，则应选择掩码最长的一条路由。路由器应该按掩码长度从长到短排序。查找路由表时，则从掩码最长的路由开始进行搜索。缺少路由的掩码长度为 0，所以应该最后进行比较。

路由表的查找过程如图 2-24 所示，其中 G 表示网关字段中的地址是一个有效路由器的地址。

4．IP 报文转发及分段

（1）报文转发

路由器收到一个需要转发的报文后，将如何进行处理，下面通过举例加以说明。

【例 2-1】如图 2-23 所示，设 R1 收到主机 A 发送给主机 C 的 IP 包。R1 的数据链路层根据帧中的以太网类型确定帧中数据是一个 IP 报文，于是交给 IP 处理。IP 首先要检验 IP 头的各个

字段的正确性，包括版本号、检验和长度等。如果有错，则丢弃该数据包；如果正确无误，则TTL 字段的值减 1。

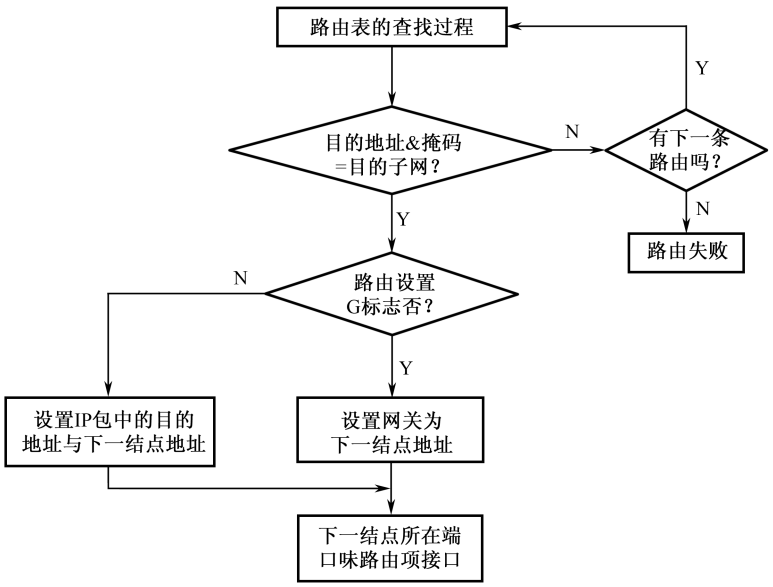


图 2-24　路由表查找

如果 TTL 的值为 0，则表明该数据包在网中的生存时间到期，应该丢弃；如果 TTL 大于 0，则根据包中目的地址寻找路由。如果没有合适的路由，则丢弃该数据包；如果找到路由，则向下一站地址转发。在转发前首先要得到下一站的物理地址，进行帧封装，然后发送出去。

在路由器中，路由器修改了 IP 头中的 TTL 字段，所以要重新计算 IP 头中的检验和。如果 IP 包带有 IP 选项，则还要根据选项的要求进行处理。

在处理的过程中，凡是出现错误、路径不通等情况，IP 都要向报文的源端发送 ICMP 报文，报告不能转发的原因。

（2）报文分段

IP 报文要交给数据链路层封装之后才能发送。理想情况下，每一个 IP 报文正好放在一个物理帧中发送，这样可以使网络传输的效率更高。实际的网络所支持的最大帧长各不相同。例如，以太网的帧中最多可以容纳 1500 字节的数据，FDDI 帧中可以容纳 4470 字节的数据。为了能把一个 IP 报文放在不同的物理帧中，最大 IP 报文长度就只能等于所有物理网络 MTU 的最小值，这在很多情况下会导致很低的传输效率。

IP 采取了另外一种方法。在发送 IP 报文时，一般选择一个合适的初始长度，如果这个报文要经历 MTU 比 IP 报文长度小的网络，则 IP 把这个报文的数据部分分割成较小的数据片，组成较小的报文，然后放到物理帧中。

分割或分片一般在路由器上进行。如果路由器从一个网络接口收到了一个 IP 报文，要向另外一个 MTU 比 IP 报文长度小的网络发送，则要把该 IP 报文分成多个 IP 分段发送。

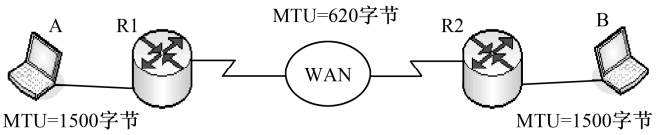


图 2-25　网络中的 IP 报文分段

【例 2-2】图 2-25 是一个对 IP 报文进行分段的网络环境示例。在图 2-25 中，两个以太网通过一个广域网互连起来。以太网的 MTU 都是 1500 字节，但是中间的广域网的 MTU 为 620 字节。如果 A 发送给 B 一个长度超过 620 字节的报文，R1 在收到后，就必须把该报文分成多个分段。

在进行分段时，每个数据片的长度依照物理网络的 MTU 确定。由于 IP 报文头中的偏移必须是 8 的整数倍，因此要求每个分段的长度必须是 8 的整数倍（最后一个分段除外，它可能比前面几个分段的长度小，长度可以为任意值）。图 2-26 为包含了 1400 字节数据的 IP 报文经过图 2-25 所示网络环境中路由器 R1 后报文的分段情况。

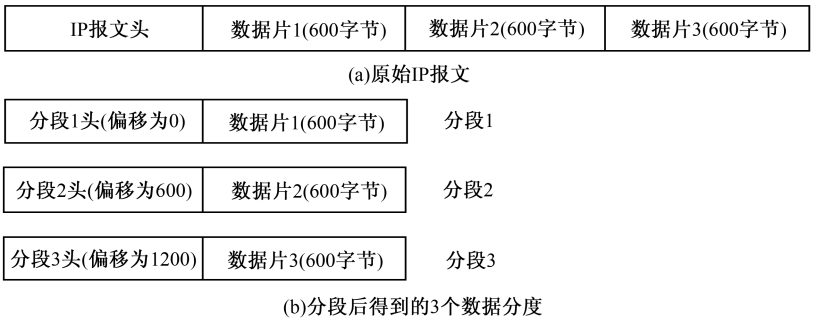


图 2-26　IP 报文分段示例

每个分段都包括一个 IP 报文头。该报文头和原始的 IP 报文头除了分段偏移、MF 标志位和检验几个字段外，其他都是一样的。

重装是分段的逆过程。在目的端接收一个 IP 报文时，可以根据其分段偏移和 MF 标志位判断是否为一个分段。如果 MF 为 0，并且分段偏移为 0，则表明这是一个完整的报文；否则，如果分段偏移不为 0，或者 MF 标志位为 1，则表明它是一个分段，这时目的端要实行分段重装。IP 根据 IP 报头中的标识符来确定哪个分段属于同一个原始报文，根据分段偏移来确定分段在报文中的位置。如果一个报文的所有分段都到达了目的地，则应将它重装成一个完整的报文交给上层协议。

5．ICMP

图 2-27 给出了 ICMP 报文的名称和分类。因为 IPv4 不支持流量控制和错误控制，也缺乏对网络的测试和诊断功能，所以由 ICMP 来弥补 IPv4 的这些不足。ICMP 的报文分为差错报告和查询报告两类。查询报文总是成对出现，一个是请求报文，另一个是应答报文，每种类型的报文用一个类型值加以区分。每个差错报告报文有一个类型值，每个查询报文有一对类型值（一

个用于查询，另一个用于响应）。

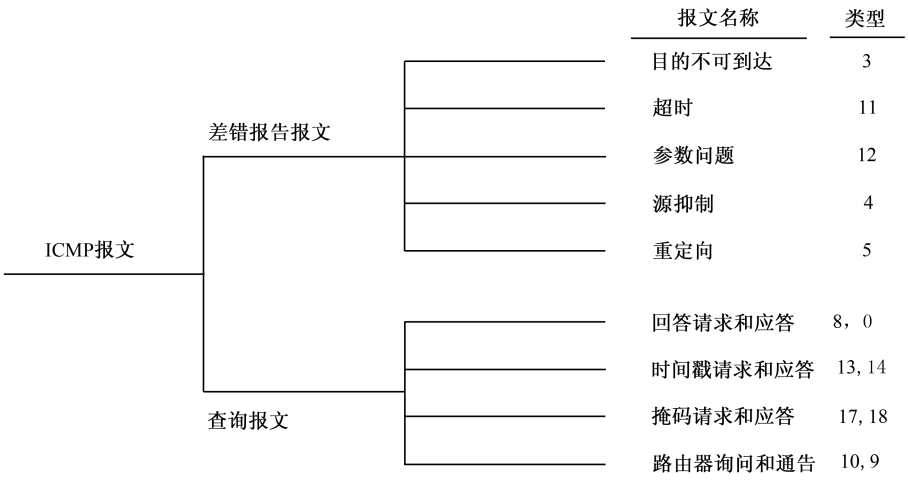


图 2-27　ICMP 报文的名称和分类

（1）报文格式

ICMP 报文的格式如图 2-28 所示。其由 8 字节报文头和可变长的数据组成。在 8 字节的报文头中，前 4 字节是所有报文共同的结构，后面的 4 字节（头的其余部分）对于不同的报文有不同的结构和定义。报文头的第 1 个字段是类型，占 1 字节，定义了报文的类型，每个类型给定一个或两个类型值，如目的不可到达报文有一个类型值 3，掩码请求和应答报文有两个类型值 17 和 18。代码字段也占 1 字节，提供关于报文的进一步信息。例如，目的不可达报文总的功能是报告主机的分组无法到达目的地，它可进一步分为网络不可到达、主机不可到达、端口不可到达等不同情况，这个不可到达的细节可以用代码值来表示。检查和占 2 字节，提供对整个报文的检查。头的其余 4 字节因报文不同而异。数据部分对于差错报告报文，它是出错分组的协议头和该分组的前 8 字节数据。对于查询报文，数据部分会因查询报文的不同有一些额外的信息，如时间戳请求报文，该报文的发送者必须在数据部分写入此时的时间戳值。

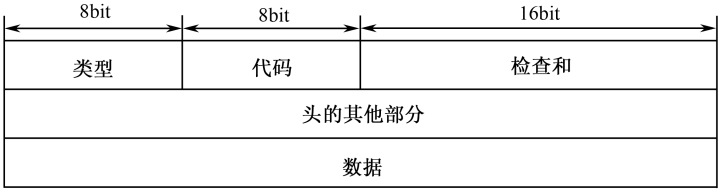


图 2-28　ICMP 报文的格式

（2）差错报告报文

IPv4 不但不能纠正错误，甚至连发现错误的功能都没有。但是，数据分组在传送过程中，丢包和包传输出错的情况总是不可避免的。每当这种情况发生时，ICMP 可以产生一个差错报告的报文传送到发送这个包的主机，由其上层（TCP）来纠正这个错误，IP 层没有检错重发的

功能。ICMP 也只是报告源主机“这个包出错了”，ICMP 自身没有纠正错误的功能。图 2-29 说明了差错报告报文的生成过程。

图 2-29　差错报告报文的生成过程

当 ICMP 要将一个出错的分组报告源主机时，它把接收分组的 IP 头和它的数据部分的前 8 字节一起作为 ICMP 报文的数据，然后加上 8 字节的 ICMP 报文头构成一个 ICMP 报文。这个报文包含原来分组的 IP 头，是为了让源主机知道是哪个出错分组的信息（根据分组的标识符字段值判断），后面的 8 字节含有 TCP、UDP 的端口号及 TCP 的顺序号，以便处理错误。这个 ICMP 报文由产生这个报告的目的主机或路由器通过 IP 分组发送到源主机。下面对 5 个差错报告报文分别给予说明。

1）目的不可到达。当路由器不能路由一分组，如路由表出错或主机不能递交分组等，此时路由器或主机就向发送这个分组的主机发一个“目的不可到达”的报文。

2）源抑制。IP 没有在源主机和路由器及源主机与目的主机之间建立反馈机制，当源主机以过多或过快的数据输入网络时，会造成路由器来不及转发，或目的主机来不及处理，也就是说，分组在路由器的输入/输出端口的缓冲区产生溢出，同样的情况也会在目的主机的 IP 层缓冲区发生，上述情况称为网络拥塞，它会造成丢包。在数据链路层是用发送窗口的大小来限制输入到网络层中的流量来解决的，这种流量控制在两个节点之间，采用接收端给发送端反馈信息的方法来解决，而在此，流量控制在网络层进行。方法都是要求发送端减少发送的数据量，源抑制报文就是为此目的而设定的。每当网络发生拥塞、大量丢包时，路由器或目的主机就向源主机发送源抑制报文，告诉它“这个包被迫丢弃了，你应该减少发送数据量”。

3）超时。一个分组不能久留在网络中，为此在发送分组的头部放一个存活时间值。在分组传送中，每经过一个路由器，就将其值减 1，减到 0 时就抛弃该分组。此时，抛弃分组的路由器就要向源主机发一个超时报文。另外一种情况是在分组分段的情况下，为各个段设定一个定时器，在给定时间内若不是所有的段都到达目的端，就抛弃这些分段的分组，并发超时报文。

4）参数问题。路由器或目的主机发现 IP 分组协议头有模糊不清之处（如协议版本不对或者协议字段值是一个不支持的协议等）时，路由器或目的主机也应抛弃该分组，并发送参数问题报文给源主机。

5）重定向。路由器转发分组依据的是路由表，由于网络的拓扑处于不断变化的状态之中，路由器不是一成不变的，因此要求路由表是动态的，它要不断收集关于网络拓扑结构的信息，及时修改路由表，使得每一个源主机到目的主机的分组路由都是最佳的。路由器修改了路由表后，要在路由器之间产生很大的路由信息交换，这是路由器的重要任务之一，它占用了路由器

很大的开销。主机也有路由表，不过为了减小主机的开销，把主机的路由表做成静态的，主机不参加路由表的修改。为了更便捷，主机的初始路由表很简单。TCP/IP 采用的原则：假定路由器知道正确的路由，而主机启动时只知道最少的路由信息，启动后在数据传输的过程中不断从路由器获得新的路由信息。从路由器获取信息来修改主机路由表的工作由 ICMP 的重定向报文来承担。在图 2-30 中，假定主机发送一份 IP 数据报给 R1。这种选路决策经常发生，因为 R1 是该主机的默认路由。R1 收到数据报并且检查它的路由表，发现 R2 是发送该数据报的下一站。当它把数据报发送给 R2 时，R1 检测到它正在发送的接口与数据报到达接口是相同的（主机和两个路由器所在的 LAN），这样就为路由器发送重定向报文给原始发送端提供了线索。R1 发送一份 ICMP 重定向报文给主机，告诉它以后把数据报发送给 R2 而不是 R1。

（3）查询报文

查询报文用来对网络进行测试和诊断，以发现网络的问题。它采用的方式是一个节点发送一个询问，要求被询问的节点做出回答，所以这类报文总是成对出现。它把要询问的内容和回答的结果作为 ICMP 报文的数据，由 IP 来传送。图 2-31 是 ICMP 查询报文的封装，可以看出，它不需要原来分组的协议头部和前 8 字节数据。查询报文有 4 对，下面分别加以说明。

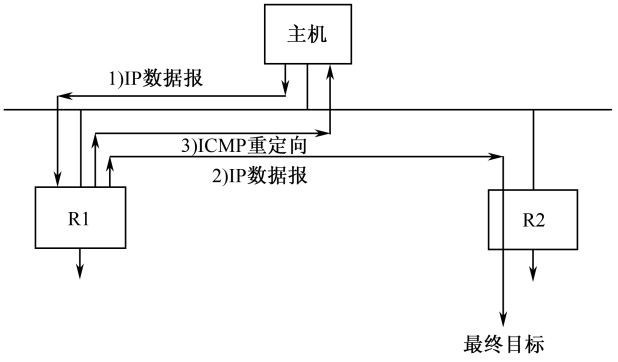


图 2-30　重定向的概念

图 2-31　ICMP 查询报文的封装

1）回送请求和应答。这个报文用来测试两个系统（主机或路由器）是否有通信能力。实际上就是请求方发送 IP 分组给应答方，应答方能够给予回答，证明两者之间的通信正常，如果在规定时间内不能收到对方的应答，说明两系统之间不能通信。Internet 的 ping 命令就是基于这个报文原理。

2）时间戳请求和应答。有时候需要知道分组在两个机器（主机或路由器）之间传输花费的时间。这时发送请求的一方在发送的报文中打上此刻的时间戳 t_1，发送到对方，应答方打上接收到的时间戳 t_2 和发出的时间戳 t_3，并将这些数据在应答报文中发回来，设收到应答的时间为 t_4，于是可知：t_3-t_2 是应答方处理报文的时间；$t_4-t_1-(t_3-t_2)$是报文在两节点之间往返传送的时间。

3）地址掩码请求和应答。一台主机可以知道它的 IP 地址，但是可能不知道它的掩码。例如，一台主机知道它的地址是 159.31.17.24，但不知道它的掩码是/24。不知道掩码就不知道网络地址，这时主机可以向路由器发送掩码请求报文，从路由器得到它的掩码。

4）路由器询问和通告。主机往往需要知道某些路由器的工作状态，它可以用广播方式发送

路由器询问报文，路由器收到这个报文以后，用路由器通告报文广播它的路由信息。有时路由器即使没有收到询问报文，也可以定期发送路由器通告报文。一个路由器的通告报文不仅说明了自己的存在，而且还说明了它所知道的其他路由器的存在。

6．IGMP

IGMP 是用来进行多播的。有许多应用需要多播来支持。例如，给多个用户发布邮件、新闻或视频会议等。

多播与同时向多个目的站发送数据是有区别的，前者的数据报仅在传送路径必须分岔时才将数据报复制后继续转发；后者则在一开始，源站就要发送多个数据报，分别传送给多个目的站。这样看来，多播可明显节约网络的资源。IP 多播实际上只是硬件多播的一种抽象，通常将能够运行多播协议的路由器称为多播路由器。

IP 使用 D 类地址支持多播，每一个 D 类地址标识一组主机。D 类地址可用来标识各个主机组（Host Group），共有 28bit，因此可以标识超过 2 亿 5000 万个组。当某一进程向一个 D 类地址发送数据报时，就是向该组中的每一个主机发送同样的数据报，但都是“尽力交付”，而某些组内成员可能收不到这个数据报。

D 类地址分成两类永久组地址和临时组地址。永久组地址不需要每次都建立组。下面是由 Internet 指派号机构（Internet Assigned Numbers Authority，IANA）所分配的几个永久组地址的例子。

- 224.0.0.1 表示在同一个局域网上的所有系统。
- 224.0.0.2 表示在同一个局域网上的所有路由器。
- 224.0.0.5 表示在同一个局域网上的所有开放最短路径优先（OSPF）路由器。
- 224.0.0.9 表示在同一个局域网上的所有路由器信息协议 RIPv2 路由器。

临时组地址可动态分配使用，如果组中没有成员，则该组将不存在。例如，如果一个公司内临时要开一个视频会议，它需要分配一个组地址，然后所有的成员都加入这个组，开始通信。在会议结束后，所有成员都退出这个组，此时这个组地址又可以被其他应用使用。

IP 对多播数据尽力发送，与 IP 点对点数据一样，IP 同样不能保证多播数据完整无误地到达组中的各个成员。

如果一个组的成员处于不同的子网上，则多播需要路由器的转发，多播的数据沿着从源端到各个组员的最短路径树传播。IP 主机在加入某一个之后，要向其相邻的路由器进行报告，这样路由器才能把所有属于该组的信息往该主机发送，IP 主机使用 IGMP 向与其直接相连的路由器报告其所属的组的地址。IGMP 与 ICMP 相似，也是 IP 的一部分，IGMP 包放在 IP 包的数据部分，协议类型为 2。IGMP 包的格式如图 2-32 所示。

版本号	类型	未用	校验和
组地址			

图 2-32　IGMP 包格式

IGMP 的当前版本号为 1。在 IGMP 中定义了两类信息。类型为 1 是主机成员询问包，由路

由器发出。类型为 2 是主机成员报告消息，它由主机发出，是对询问包的响应。校验和字段对 IGMP 包进行校验。在询问包中，组地址字段为 0，在报告包中，则存放着被报告的组的 IP 地址。报告包的目的地址是被报告的组地址，并且 IP 包头的 TTL 为 1。

多播路由器定期向全主机组地址 224.0.0.1 发送主机成员询问信息，以发现在其所连接的局域网上存在的主机组。应该说明的是，多播路由器只关心在局域网上到底有哪些成员组，而对于网络上到底哪台主机属于哪个组并不关心。主机在收到询问后，则会产生主机成员报告发送回去，报告它所属的主机组。为了防止多台主机同时发送报告消息产生冲突，并减少总的报告数，IGMP 协议做了如下规定。

- 当主机收到询问后，并不是马上发送报告，而是为它所属的每一个组地址启动一个时钟，每个时钟的时限是在 0～10 秒（IGMP 的默认值为 10 秒）间的一段随机的时间。如果某个超时，则为相应的组产生报告包并发送出去。
- 如果主机收到了其他主机发送的报告包，则停止相应组的时钟。在通常情况下，每个组仅产生一个报告信息。

此外，如果在时钟等待期间，又收到另一个询问包，并不必刷新每个时钟的时限，而是在源时钟超时时则发送报告。在一台主机新加入一个多播组后，应该马上发送组报告，而不用等待询问消息，因为该主机可能是该组的第一个成员，这些都可使路由器尽快获得最新的多播组信息。多播数据报在传输的过程中，若遇到有不运行多播软件的路由器或网络，那么就采用隧道（Tunneling）技术。

例如，图 2-33 表示隧道技术在多播中的应用。网 A 中的主机 A 向网 B 中的 B、C、D、E 目的主机进行多播。但路由器 R1 和 R2 并不运行多播软件，因而不能按多播地址转发数据报。为了解决这个矛盾，路由器 R1 就对多播数据报进行再次封装，即加上一个普通数据报的首部，使之成为一个向单一目的站发送的单播数据报，然后通过“隧道”从 R1 发送到 R2。单播数据报到达路由器 R2 后，再由路由器 R2 剥去其首部，使它又恢复成原来的多播数据报，继续向 B、C、D、E 目的主机转发。

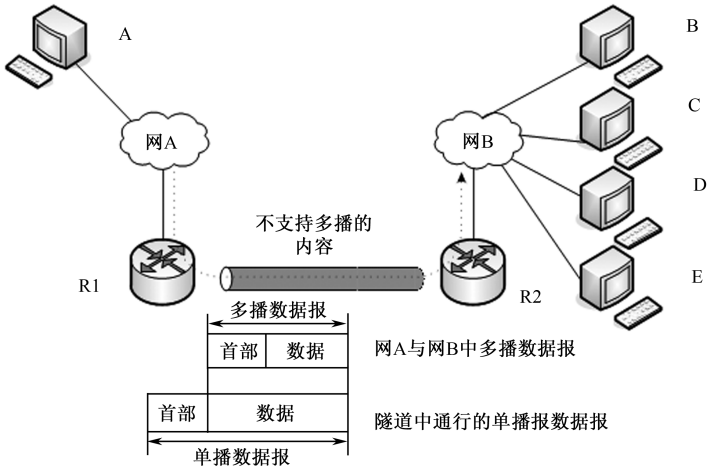


图 2-33　基于隧道技术的多播

2.2.2　路由协议

本节将介绍分组在进行转发的过程中选择合适路径的几种典型路由选择方法，实施路由选择方法功能是由分组转发过程中的中间节点设备——路由器来完成的，而且路由器能够实现其中一种或多种路由选择方法。路由器进行分组转发的依据是路由表，静态路由选择方式中，路由器按照固定的方式转发分组，路由表是固定的，没有自动构建路由以适应网络变化的能力。动态路由方式中，路由具有自学习能力，能自动采集和交换路由信息，自动构建路由表，并能适应网络的变化。大多数路由器使用的是动态路由方式，其中路由器采集和交换路由信息则使用相关的路由协议来实现。

1．路由器和路由协议

路由器（Router）是网络层非常重要的网络互连设备，因此分组转发过程中的主要中间节点设备就是路由器。在 Internet 的传统定义，路由器也被称为网关（Gateway）。

路由器的主要功能是进行路由选择和分组转发。互联网络中的路由选择和单个网络中的路由选择类似，只是要复杂得多。当一个网络中的主机要给另外一个网络中的主机发送分组时，它首先把分组送给同一个网络中连接的路由器，路由器读取分组的网络层协议头，根据其目的地址信息（包括网络号和主机号），选择合适的路由，把该分组传递到下一个路由器或终端的目的主机。图 2-34 所示为路由器连接不同网络（其中每条边实际代表一个网络）实现的一个互联网络，对于从 H1 到 H4 的分组，将会沿着 R4、R5、R3 的路径转发。

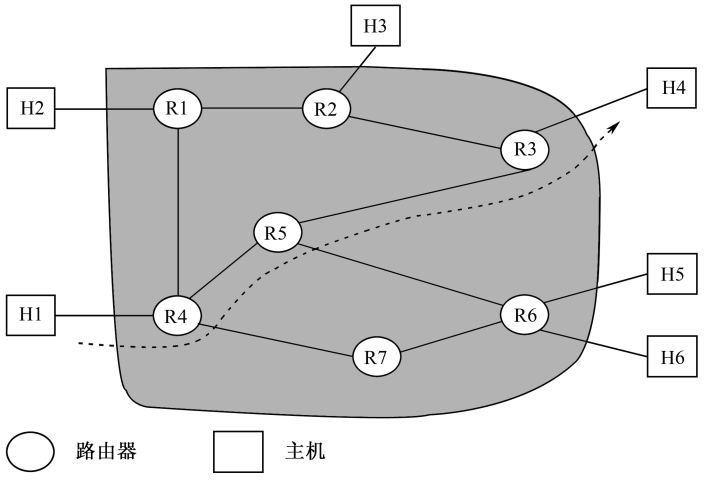


图 2-34　路由器互连的一个网络

路由器是 OSI 模型的第三层设备，这意味着它实现了 3 层次的功能。下面从 TCP/IP 体系结构上，将这 3 层次进行说明，即合并物理层和数据链路层，将网络分为协议和互连两个子层。

（1）接口级

如同任何连接网络的设备，路由器确保连入传输介质的物理接口，包括协调电路信令等级、

线路和逻辑编码、为路由器配备特定类型的连接器。在路由器中的不同模型经常提供不同集合的物理接口以组合连接 LAN 和 WAN 的各种端口。特定数据链路层协议与其接口不可分割地联系在一起，如以太网、令牌环或 FDDI，连接到 WAN 的接口通常只决定某些物理层标准，基于这些标准，多个数据链路层协议可以在一个路由器中操作。例如，一个有 4 个端口的路由器实施了下列物理层和数据链路层接口：两个以太网 10 BaseT 和 10 Base2，令牌环的 UTP，LAP-B、LAP-D、LAP-F 协议可以操作的 V.35（可连接到 X.25、ISDN 或帧中继网络）。

总之，路由器接口执行整套与帧传输相关的物理层和数据链路层功能，包括访问介质（如果需要的话）、形成信令、接收帧、计算校验和，以及在校验和值正确时传递帧数据字段（剥离数据链路层头部后）到上层网络层协议实体。

（2）网络层协议

路由器检查分组的网络层头部并且更新该头部的一些内容。首先，需要测试校验和，如果分组受损，必须将之丢弃；然后再检查分组花费在网络中的时间是否超出了允许的生命周期（TTL），如果超出了 TTL，也将分组丢弃；最后对头部的一些字段进行更新，如修改 TTL 值并重新计算它的校验和。

路由器最重要的转发（Forwarding）功能就是由路由器的网络层来执行的。网络层协议实体利用从分组头部检索出的网络号找到路由表中的某一行，其中包含下一个路由器的网络地址及需要传递分组以确保它向正确方向传递的端口号。在下一个路由器的网络地址被传递到数据链路层之前，它必须利用 ARP 获取下一个路由器的 MAC 地址。从网络层，下一个路由器的 MAC 地址和路由器的端口号被传递到数据链路层。基于特定的端口号，分组被传给其中一个路由器接口，分组然后以恰当形式的帧封装起来，下一个路由器的 MAC 地址被放置在帧头部的目的地址字段中，随后，将帧发送给网络。

路由器另一个重要的过滤（Filtering）功能也由路由器的网络层执行。通常，路由器除了能够分析网络层分组的结构外，也能够分析传输层消息的结构。这样其过滤器可以阻止某些主机（源端或目的端）的分组或某些应用服务的消息（如 Telnet 服务）进入或离开网络，这种过滤通过分析主机地址或传输层协议类型字段来执行。

此外，路由器软件一般配备高级 GUI 管理工具，使得管理员可以正确地制定复杂的转发和过滤规则。

（3）路由协议层

路由器的网络层协议在实现转发的操作过程中积极地使用了路由表。然而，这些并不包含创建或者维护它的内容。这些功能由路由协议完成，基于这些协议，路由器交换网络的拓扑信息，然后分析收到的数据来判断满足特定条件的最佳路由，分析的结果构成了路由表的内容。除了前面列出的功能，也可以为路由器指派其他功能，如分段与重装。

路由器从概念上可分为主干路由器（Backbone Router，旨在构建通信载体或大型 AS 级核心网络）和边界路由器（Edge Router，连接骨干和周围网络）。二者一个侧重于转发，一个侧重于过滤。根据其他划分方法，也可将路由器分为载体路由器、企业路由器、部门路由器、远程办公路由器、软件路由器等。

从路由器的介绍中可知，路由器能够采集和交换路由信息，然后根据所掌握的路由信息，

路由器使用相应路由方式或路由算法，一直努力地构建和维护其用于分组转发的路由表。在图 2-34 中，各路由器间经过一段时间的路由信息交换后，就会在自己的路由表中建立好到达 H4 的最佳路径，即 R4 收到 H1 的分组后，就会根据自己的路由表转发给 R5，而 R5 则转发给 R3，R3 则转发给目的主机 H4。

2．路由信息协议

路由信息协议（Routing Information Protocol，RIP）是一个简单的距离向量路由协议。RIP 可以在主机或路由器中实现，因此 RIP 被分为两种不同类型的操作方式。主机中实现的 RIP 工作在被动状态，它不会传递自己的路由表中的信息给其他路由器，它只是静静地倾听其他 RIP 路由器广播的路由信息，并且根据收到的路由信息更新自己的路由表。路由器中实现的 RIP 工作在主动状态，它定期把路由信息传递给其他 RIP 路由器，并且根据收到的 RIP 消息来更新自己的路由表。也就是说，被动 RIP 的操作相对来说简单、直接得多。

每个 RIP 路由器都保持了一张路由表，每一项对应着一个目的地，其中每项包括了目的地的 IP 地址、到目的地的路径的距离度量、到目的地的路径的下一个路由器 IP 地址（如果目的地是直接连接的，不需要这个字段）、路由改变标志（指示这条路由信息是否最近被改变过）及与这条路由有关的一些计时器。RIP 采用的距离度量是一种非常简单的测量到目的地距离的方式：站点计数度量。路由器把它直接连接的网络的距离定义为 1，如果距离为 n，表示它到达目的地途中要经过 n 个路由器，即距离给出了该路由要经过的路由器的个数。为了能够应付一些复杂的情况，如经过一个速率较高的网络的路由器显然比一个慢速网要更好，RIP 在具体实现时常常允许管理人员对这些慢速的网络指定一个更大的距离度量值（如 3）。当路由器初始化时，它会把那些到达它所直接连接的网络的路由加载进来，到直接连接的网络的距离一般被设置为 1。一般 RIP 的具体实现也允许管理人员增加新的路由，如不是通过 RIP 了解到的路由。

每个 RIP 路由器每隔 30 秒广播一次路由信息，RIP 路由器也可能通过发送 Request 消息来询问其他路由器有关某些路由器或者所有路由的信息。例如，当一个主机启动后，可能要求相邻的 RIP 路由器传递路由表中的所有信息。当一个 RIP 路由器从邻居路由表收到路由消息时，它对消息中包含的到某个目的地的路由信息运用距离向量路由算法，以决定是否找到一条经过该邻居路由器到目的地新的、距离更短的路径，若找到，则更改其路由表中那条目的地的路由。若路由消息中新通知的路由和原来的路由距离相同，则 RIP 仍然选择使用老的路由，这有助于保持路由的稳定。但是如果某个路由器崩溃或者某一网络连接出现了故障，这条路由就不会继续存在，因为 RIP 在路由表中对每条路由都有一个计时器，当收到新的有关这条路由的消息时，该计时器被重新设置，如果计时器超时（通常为 180 秒，即 n 次中有 k 次机制：路由器每连续 6 次中只要收到一个定期广播的路由消息，就可以认为该路由合法，这样在一定程度上防止了由于路由消息的丢失而带来的不可靠性），这条路由就被宣告为非法，即目的地不可达。注意它并不是马上从路由表中删去，因为这条失效的路由还应该向邻居路由器报告，经过一段超时（被称为 Garbage-collection Timer）后，该路由最终被路由器去除。

怎样把某条路由宣告为不可达呢？RIP 通过选择一个特殊的距离度量值，该值大于所有正常路由的距离，在目前的实现中，这个值被选为 16，它一般被称为“无穷大”，因为它大于任

何合法的距离度量。实际上 RIP 无穷大的取值还综合考虑了网络规模和收敛速度两方面。

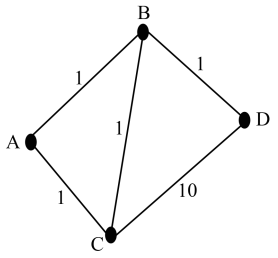


图 2-35　一个简单的互联网配置

（1）处理无穷计数问题

和其他距离向量路由协议一样，RIP 当然也会遇到无穷计数问题，造成无穷计数的原因是它对好消息的传播快，而对坏消息的传播则很慢。为了更好地描述 RIP 解决这个问题的机制，考虑图 2-35 的配置，除了 CD 之间的距离为 10 外，其他链路的距离都为 1。为了简单起见，只考虑连在 D 上的目的网络的路由，并假设 B、D 之间的链路现在出现故障。表 2-4 给出了路由交换的过程，表中每一项表示到目的网络的下一个路由器跳段的地址和距离，其中假设所有路由器同时发送 RIP 路由消息。

表 2-4　BD 线路断开后的路由交换过程

D	Dir,1	Dir,1	Dir,1	Dir,1	Dir,1	…	Dir,1	Dir,1
B	D,2	Unreach	C,4	C,5	C,6	…	C,11	C,12
C	B,3	B,3	A,4	A,5	A,6	…	A,11	D,11
A	B,3	B,3	C,4	C,5	C,6	…	C,11	C,12

注：Dir 表示直接连接，Unreach 表示不可达。

路由器 B 可以使用超时机制把那条失效的路由去除，但是该链路故障的影响会在网络中持续一段相当长的时间。起初 A 和 C 都认为可以通过 B 到达 D，所以它们发送到 D 的距离为 3 的路由消息。接着 B 将认为可以通过 A 或者 C 到达 D，而事实上，这当然是不可能的。A 和 C 宣称的到 D 的路由已经不存在，但是它们还不知道这一点。甚至当它们发现那条通过 B 到 D 的路由消失时，它们会互相认为有一条通过对方到达 D 的路由。最坏的情况是，如果某个部分完全从网络中断时，到这部分网络的距离会慢慢地每次一点一点地增加，直到它们最后到无穷大。

从上面可以看到，无穷大的值取得越小越好，如果网络中某个部分不能访问，路由算法可以更早地收敛，同时无穷大应该足够大，使得正常路由的距离不会超过它。所以无穷大的取值是对网络规模和收敛速度的平衡，RIP 选择无穷大为 16。

为了解决无穷计数问题，提出了许多机制。其中一种是抑制（Hold-down）规则，一旦路由器了解到某个网络不可达（距离为无穷大），在一段时间（抑制期）内忽略所有有关那个网络的路由信息。抑制期必须足够大，以使得网络的不可达状态能够在这期间传播所有其他的路由器，一般被设定为 60 秒。值得注意的是，所有路由器都必须使用相同的抑制期，否则有可能发生路由回路。抑制规则的缺点是一旦出现路由回路，它将至少在抑制期间持续下去。更重要的是在抑制期内，即便可能有其他路由到达那个网络，那些可能不合适的路由都会一直使用。

（2）改进水平分割

在前面的例子中，无穷计数问题是因为 A 和 C 之间有一个路由回路，它们都认为通过对方可以到达 D。实际上，如果把从它的邻居了解到的到某个目的网络的路由再传递给该邻居是毫无意义的。这也正是水平分割机制的基本思想，即通过一个特定的网络接口发送 RIP 更新消息时，绝对不要包括通过那个网络接口学习到的路由信息。RIP 还对水平分割进行改进，这就是毒性反

转的水平分割（Split Horizon With Poisoned Reverse）。路由器并不是不给邻居路由器发送通过该邻居了解的路由信息，而是和往常一样，给邻居路由器发送路由信息，只是那些从该邻居了解的路由信息的距离被设置为无穷大，以致到目的网络不可达。

现在尽管 A 认为可以通过 C 到 D，A 送给路由器 C 的消息指示 D 为不可达。如果通过 C 的确有一条路由到达目的地，那么 C 或者有一条到 D 的直接连接，或者有一条通过其他路由器到达 D 的路由。路由 C 到 D 的路由无须经过 A，否则将会有一个路由回路。A 明确地通知 C 到 D 是不可达的，以免 C 怀疑有一条通过 A 到达 D 的路由。

一般来说，毒性反转的水平分割比简单的水平分割要更好。如果两个路由器相互之间有一个路由回路，即每个路由器都认为有一条通过对方到达目的地的路由，通过通知对方到目的地的路由的距离为无穷大，可以马上解除这个回路。而如果不通知，不正确的路由必须等待一段时间才能取消。但是毒性反转的水平分割增加了路由信息的大小，从而要使用更多的网络带宽。在大多数情况下，这可能不是一个很大的问题。

（3）触发更新

毒性反转的水平分割只是防止发生在两个路由器之间的路由回路。但是仍然会出现 3 个或多个路由器之间有一个路由回路的情况。例如，A 可能相信它有一条通过 B 到达目的网络的路由，B 相信有一条通过 C 到达目的网络的路由，C 相信有一条通过 A 到达目的网络的路由。水平分割无法解除这个回路，因为对于每一对邻居路由器，它们在传递到目的地的路由消息时只是单向的，而不会往回传，只有等待路由算法逐渐收敛，最后到目的网络的距离为无穷大，从而被认为不可达。触发更新有助于加快这个收敛过程，它的思想非常简单，只要增加一条规则：一旦路由器到目的网络的距离改变，马上发送一个路由更新消息，而不管是否到了定期发送路由更新消息的时候。

假设一个路由器有一条通过路由器 G 到目的地 N 的路由，如果路由器 G 到目的地 N 的距离改变（如 G 到目的地 N 是不可达的，距离为无穷大），将发送一个路由更新消息。接收到这个路由更新消息的路由器会把它和原来的路由比较，如果距离改变，该路由器将继续按照触发更新规则继续给直接相连的主机和路由器发送路由消息，那些路由器再继续传递给它们的邻居路由器。这样就导致一系列串联的触发更新，使得路由消息的变化能够更快地传播。事实上如果一系列的触发更新足够快，即在这一系列触发更新完成之前系统保持静止（没有定期发送正常更新消息的时间），可以证明无穷计数问题将再也不会出现，坏路由会马上移走，从而不会形成任何路由回路。

RIP 规定在传输触发更新消息时，必须延迟一段随机的时间再发送，即当一个路由器的某条路由的距离改变时，等待一段非常短的随机时间后，马上发送路由更新消息，而不需等待到达定期发送正常更新消息的时刻才发送，这种机制可以防止触发更新消息产生过多的网络负载。假设一个以太网上有多个路由器，路由器通过广播方式来传递路由信息，当其中一个路由器中到某个目的地的距离变化时，发送一个触发广播消息，以太网上的所有其他路由器都将收到这个消息，并注意到路由的距离改变了，从而这些路由器可能同时广播它们的触发更新，这样导致了一个广播风暴的产生。经过等待一段很小的随机延时再发送触发更新，可以避免很多路由器同时传播触发更新的情况。

（4）RIP 消息

RIP 是一个基于 UDP 的协议，RIP 消息是通过 UDP 服务来发送的，所使用的 UDP 端口号为 520。RIP 消息可以分为两类：请求路由信息消息（RIP 消息的 COMMAND 字段为 1）和路由信息消息（RIP 消息的 COMMAND 字段为 2）。一个路由器或者主机可以通过发送请求路由信息消息要求另一个路由器传递路由信息，路由器通过发送路由信息消息来响应。一个请求可以要求获得到任何目的地的路由信息，也可以指定要求获得到某个特定目的地的路由信息。路由响应消息给出了到目的地的路由情况，除了在收到请求路由消息时发送响应外，路由器还会定期（每隔 30 秒）发送路由信息消息。

RIP 消息都具有一个统一的格式，如图 2-36 所示。由图中可以看到其中没有长度字段，这是因为下层的 UDP 有封装功能，从而可以知道消息的边界，命令（COMMAND）字段指示 RIP 消息的类型（Request 或 Response）。RIP 消息格式中包括了地址家族标识（Address Family Identifier）字段，这种对路由器地址的使用方式使得 RIP 也可以在其他网络层协议下使用，而不是局限在 TCP/IP 环境中。

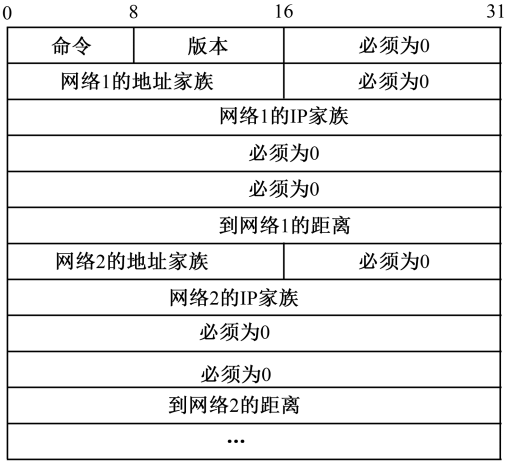


图 2-36　RIP 消息格式

RIP 有很多局限性。因为 RIP 选择 16 作为无穷大，不能用在网络直径大于 15 的网络中，同时 RIP 使用的距离度量非常简单，不能采取一种动态的方法（如根据网络延迟或负载）来选择路由，而且尽管 RIP 采用了很多措施（如毒性反转的水平分割和触发更新等）来解决无穷计数问题，但是这种可能性仍然存在，因此 RIP 一般用在网络规模不是很大的场合。

RIP 是距离向量路由算法的一个最直接的实现，它有两个版本，在早期版本（RIPv1）的设计过程中，许多问题都没有考虑到，或者许多问题都是在 RIPv1 出来后才涌现出来的。RIPv2 吸取了实际网络运行经验和用户需求，对 RIPv1 做了一些扩展，首先它支持子网的概念，支持那些通过其他外部协议（如 BGP）了解到的路由，支持 IP 组播，支持路由器之间的认证机制等。

3．开放最短路径优先协议

上面介绍了距离向量路由协议的一种：RIP，它在网络规模不是很大时能够正常工作，但如果网络规模很大，RIP 要求定期交换路由信息，可能占用过多的带宽，其运行并不令人满意，并且 RIP 还受到无穷计数问题的困扰。在这种情况下，一种新的内部网关协议被提出来，这就是开放最短路径优先协议（Open Shortest Path First，OSPF），并且成为目前主要的内部网关协议。下面将简单地介绍 OSPF 协议的基本思想，有关 OSPF 协议的详细情况，可以参考 RFC 2328。

OSPF 是一种链路状态路由协议。在链路状态路由协议中，每个路由器维护其各自的本地

链路状态信息（即路由器到子网的链路状态和可以到达的邻居路由器），并且通过扩散的办法把更新的本地链路状态信息广播给自治系统中每个路由器，这样每个路由器都知道自治系统内部的拓扑结构和链路状态信息。路由器根据这个链路状态库计算出到每个目的地的最短路径（路由），所有路由器都采用相同的算法（Dijkstra 的最短路径算法）来计算最短路由，而且这个计算是在路由器本地进行的。

OSPF 是一种动态的路由算法，能够自动且快速地适应拓扑结构的变化。每次收到一个链路状态消息，并且这个链路状态消息改变了整个自治系统的拓扑或链路状态时，都要重新计算到每个目的地的最短路由。OSPF 支持负载平衡功能，当同时有几条到目的地的最短路径时，可以将负载分流到这些路由之上。因为链路状态信息的描述非常小，并且很少需要进行传输，链路状态算法所使用的带宽非常小。

OSPF 允许网络管理人员配置路径花费的度量，每个路由器根据花费计算具有最小花费的路由，如路径花费（Cost）可以与网络延迟、数据速率、金钱或其他因素有关。

OSPF 支持区域概念，区域实际上是在 IP 子网技术的基础上产生的，是子网的一般化表示。区域可有效地减小网络带宽的浪费，且有助于网络管理人员更好地进行管理。

OSPF 支持认证服务，只有被授权的路由器才能进行自治系统的路由处理，这样就可以防止有人通过向路由器发送假路由信息来欺骗路由器。

OSPF 允许路由器交换通过其他方法（如通过 BGP）了解到的路由信息，通过显式地说明所了解到路由信息的来源，把它与通过 OSPF 协议了解到的链路状态信息分隔开来，所以路由的来源和可信度不至于被混淆。

（1）链路拓扑

OSPF 支持 3 种类型的连接和网络。

① 点到点网络：即连接一对路由器的网络，如一个 56kb/s 的串行线路。

② 广播网络：该网络支持广播功能，并且有至少两个以上的路由器连接在上面，如以太网。

③ 非广播方式的网络：该网络有多个（大于两个）路由器连接在上面，但不支持广播功能，如 X.25 分组交换网。OSPF 把非广播方式的网络进一步分为两种方式，第一种称为非广播多路访问（Non-Broadcast Multi-Access，NBMA）网络，连在这个网络上的路由器间可以直接进行通信，它和广播方式的 OSPF 运作类似。第二是点对多点（Point-to-Multipoint）网络，这种方式把非广播网络看成多个点到点链路。

在 OSPF 协议中，每个路由器维护了一个反映它了解到的、所在自治系统的拓扑链路状态库。这个拓扑可以用一个有向图表示，其中有向图中的顶点由自治系统内的路由器和网络组成，一条连接两个路由器顶点的边表示这两个路由器通过一个点到点物理链路连接，一条连接路由器顶点和网络顶点的边表示该路由器连哪个网络上，每个路由器顶点的外出边被分配一个花费，这个花费可以由网络管理人员配置。网络有两种类型，如果某个网络可以传递那些既不是从自己网络出发，又不是发送到这个网络的分组，这类网络被称为传输网络。另外一种网络被称为中断网络，传输网络在有向图中有离去边和到来边连接，而中断网络只有到来边。

图 2-37 给出了各种网络是如何用有向图表示的。对于点到点网络，通过一个连接两个路由器顶点的双向边来表示，点到点网络接口可以不分配任何 IP 地址。如果接口分配了 IP 地址，

它们可以通过一个中断连接来模拟表示，即每个路由器宣告一条到另一个路由器的地址对应的中断网络的连接。图 2-37（b）表示一个只有一个路由器连在上面的网络（即中断网络），这时网络节点是中断连接的末端。如果多个路由器连接在一个广播网络上，在途中通过每个路由器顶点到网络顶点的双边来表示，如图 2-37（c）所示。图中的每个网络都有一个 IP 地址及网络掩码，该网络掩码表示了网络中节点的个数，特别地，直接连到路由器的主机通过中断网络来表示，网络掩码为 0xffffffff，表示该中断网络只有一个节点。

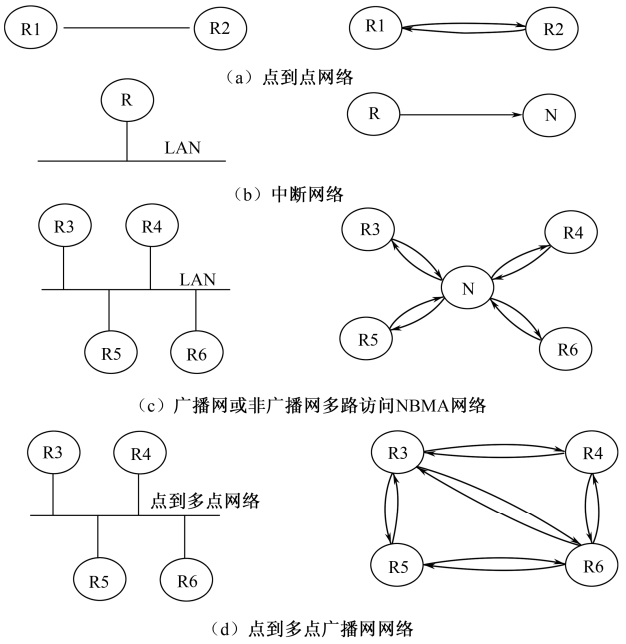


图 2-37　网络有向图表示

前面提到非广播方式的网络可以分为两种方式，相应地，它们的有向图表示也不同。对于 NBMA 网络，它的有向图表示和广播方式的网络相同，即用每个路由器和网络顶点的双向边来表示。这种方式下的链路状态库大小和路由消息负载要比下面提到的点到多点方式要好很多。但是这种方式要求连接在 NBMA 网络上的所有路由器之间能够直接通信，有些非广播网络（如使用 SVCD 的 ATM 子网）可以满足这个条件。但是有些网络可能没法满足这个限制，如只支持 PVC 的帧中继网络。为了表示这种网络，一种方法是把它分成几个逻辑子网，其中每个逻辑子网中的路由器间接能够直接连接。但是这种方法要求管理人员必须能够正确地进行配置，因此按照点到多点非广播网络方式来处理可能更好，它由多条路由器间的点到点连接组合而成。实际上在表示点到多点网络中，该网络并不在有向图的顶点中出现。在图 2-37（d）所示的点到多点非广播网络中，除了 R4 和 R5 之间不能直接通信外，所有路由器之间都可以互相通信，图中右边给出的是左边这个点到多点配置的有向图表示，图中没有标识的边的花费为 0，注意从网络到路由器的边的花费总为 0。

图 2-38 所示为一个自治系统示例。其中 H1 是一台主机，通过 SLIP 连接到路由器 R12 上。所有路由器的名称以 R 开头，而网络的名称以 N 开头，路由器之间的边表示通过点到点网络连接。

路由器 R5 和 R7 是该自治系统的边界网关，和其他自治系统之间有一条 BGP 连接，图中显示了几条通过 BGP 协议了解到的外部路由。图 2-39 给出了图 2-38 所示的自治系统的有向图表示。

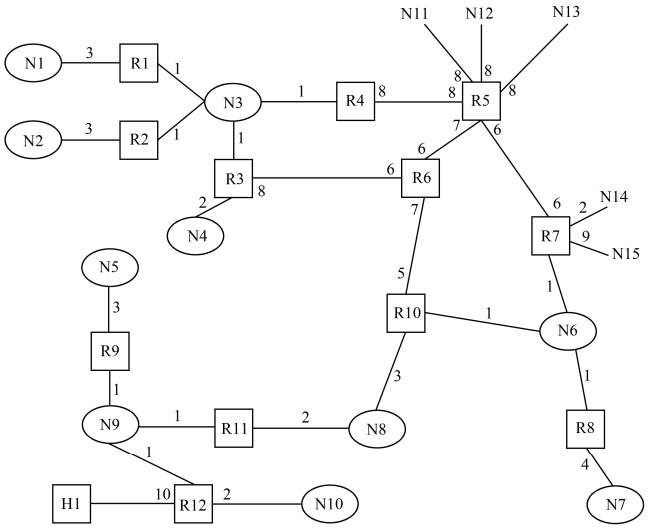


图 2-38　自治系统示例

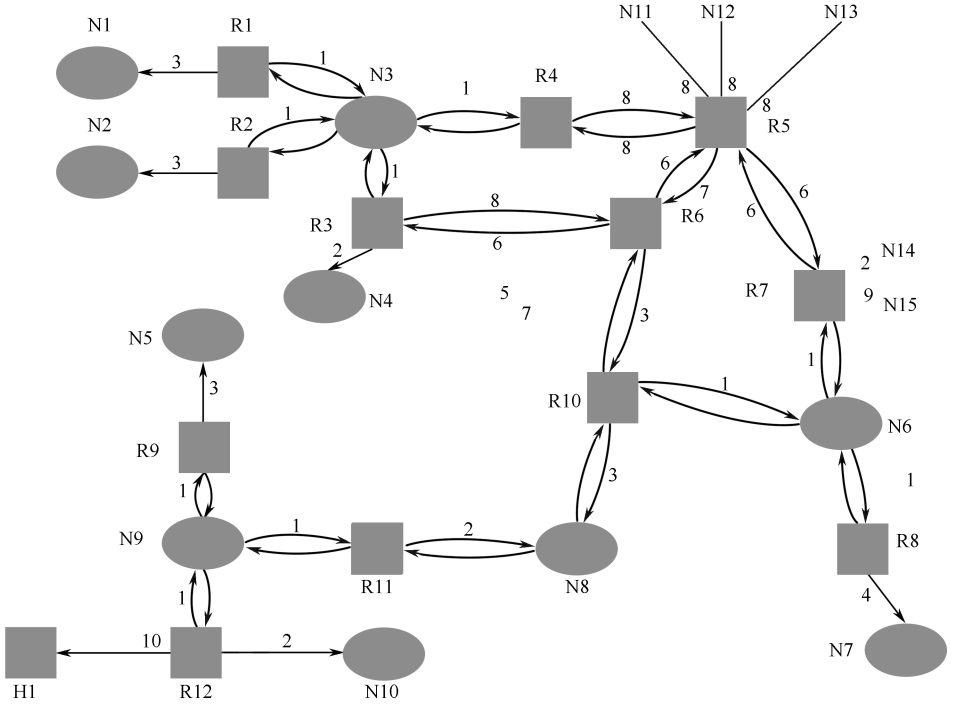


图 2-39　自治系统的有向图表示

注：图中没有标注的边花费为 0

（2）区域

由于自治系统可能越来越大、越来越难以管理，OSPF 引入了区域的概念，即把许多网络和主机组合在一起，再加上连接在这些网络上的路由器，这些合起来被称为一个区域（Area）。每个区域内部都运行一个基本链路状态路由算法，即每个区域内部有它自己的链路状态库和相应的有向图，同时运行区域内的所有路由器的链路状态库是一致的。一个同时连接多个区域的路由器运行多个链路状态路由算法（每个链路状态路由算法的副本一一对应一个区域），这种路由器被称为区域边界路由器。但是也正是由于区域的引入，整个自治系统中所有路由器的链路状态库并不一定是一致的。一个区域内的拓扑结构和细节对于自治系统的其他部分是不可见的，同样区域内的路由器也并不知道区域外拓扑的具体细节，这种方式大大地降低了路由消息所耗费的带宽。

每个自治系统都有一个主干（Backbone）区域，称为区域 0。所有区域都与主干区域连接，即主干区域包括所有的区域边界路由器，主干区域负责各个非主干区域之间的路由信息的分发。为了保证主干区域的连接性，OSPF 允许使用虚拟链路来连接两个连在一个共同的非主干区域上的主干路由器，就好像这两个路由器之间通过一个没有编号的点到点网络（即没有分配 IP 地址）连接一样。

引入区域的概念之后，自治系统内的路由被分为两类，如果网络分组的源端和目的端都在同一个区域内，只需要根据区域内部的路由信息来选择路由，这被称为区域内路由。而如果源端和目的端分别位于不同的区域，该路由被分成 3 个部分：首先是从源端到某个区域边界路由器的区域内路径，其次是源端和目的端所在区域之间的主干区域内路径，最后是从一条到目的端的区域内路径，这被称为区域间路由。从区域间路由的角度看，可以把自治系统看做一个星形配置，主干区域为中心（Hub），每个非主干区域为辐射点（Spoke）。

基于上面的描述，可以按照所要完成的功能把 OSPF 中的路由器分为以下几类，注意这些分类可能互相重叠，图 2-40 所示为 OSPF 的各种路由器。

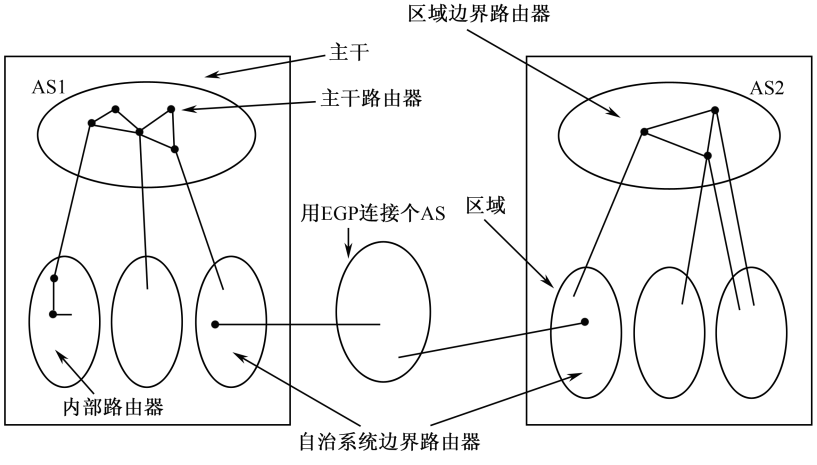


图 2-40　OSPF 的各种路由器

1）内部路由器：它所连接的所有网络都属于同一区域，该路由器只允许运行链路状态路由算法的一个副本。

2）区域边界路由器：该路由器连接在多个区域中，它运行多个链路状态算法的副本，每个区域为一个。区域边界路由器把它连接的区域的融缩后的拓扑信息传递给主干区域，再由主干区域分发给其他区域。

3）主干路由器：连在主干区域的路由器，它除了包括所有的区域边界路由器外，还可能包括其他路由器。

4）自治系统边界路由器：和其他自治系统中路由器交换路由信息。自治系统边界路由器可能是内部路由器，也可能是区域边界路由器，并且可能属于主干区域，也可能不属于主干区域。

（3）OSPF 的过程

在正常情况下，每个路由器要定期扩散链路状态更新信息。为了保证参加这个链路状态路由算法的所有路由器都能收到这个消息，路由器之间建立一个逻辑连接，并沿这个逻辑连接传递链路状态信息。路由器收到一个消息时，如果该消息是重复的，则丢弃它，否则在它的除了到来连接外的所有逻辑连接上传播该消息上的一个副本，扩散消息被确认（Link State Ack），以保证可靠性。

路由器之间的逻辑连接被称为一个邻接（Adjacency），在大多数情况下，邻接是在邻居路由器之间的连接，如一条连接两个邻居路由器的点到点链路。但是并不是所有的邻居路由器都是邻接路由器。假设所有的邻居路由器都是邻接路由器，考虑一个以太网，如果有 N 个路由器连在上面，则总共可能有 $N(N-1)/2$ 条邻接，每次当其中一个路由器收到一个消息时，它将发送一个同样的消息给每个邻接路由器。在最坏情况下，每个邻接路由器立即把收到的消息扩散给其他邻接路由器，因此这个消息在被检测到重复时，可能被传递给大多数邻接路由器两次，这样大概有消息的 N^2 个副本在该网络上传递，而实际上只要 N 个副本就足够了。为了防止这种情况，OSPF 引入了选取路由器（Designed Router）的概念，即对于广播网络或者前面提到的 NBMA 网络，连在上面的所有路由器选举一个选取路由器作为代表，它被认为与所有邻居路由器邻接，但是其他邻居路由器之间没有邻接，只有邻接的路由器之间才能交换路由信息，选取路由器负责把它所连接的网络的链路状态信息传播给其他路由器。除了选取路由器外，一般还有个备份路由器，以备选取路由器出现故障时使用。

为了发现邻居路由器及维护相应的邻接关系，OSPF 使用了 Hello 协议来定期交换消息。对于点到点物理链接来说，只要在链路上发送 Hello 消息即可。而在广播方式网络中是通过多点广播来进行的，而对于非广播网络，则可能需要网络管理人员进行配置以发现邻居路由器。

在通常情况下，每个路由器会定期扩散链路状态更新（Link State Update）消息到其邻接路由器，该消息给出了链路的状态、邻接关系及路径花费。当链路的状态、邻接关系或者路径花费有变化时，路由器也会马上发送链路状态更新消息。

数据库描述（Database Description）消息给出了发送者所拥有的所有链路状态的当前顺序号。接收者通过比较这个序号，即可知道谁的数据最新，这个消息在路由器刚加入进来时使用，以便能够拥有最新的链路状态信息。

一个路由器在和邻接路由器交换数据库描述消息后，可能会发现自己的某部分数据已经过

时，路由器可以发送链路状态请求（Link State Request）消息来要求邻接路由器传递最新的消息，表 2-5 给出了 OSPF 所使用的 5 种消息。

表 2-5　OSPF 所使用的 5 种消息

消息类型	描　述
Hello	用于发现谁是邻居
Link State Update	为邻居提供发送者的链路状态更新
Link State Ack	确认链路状态更新
Database Description	通知发送者有哪些更新
Link State Request	向对方请求链路状态信息

综上所述，路由器通过扩散把自己的链路状态信息告诉它所在区域的其他路由器。这样每个路由器都建立一个它所在区域的有向图，并计算出最短路径。主干区域中的路由器也进行这样的过程，另外主干区域还从区域边界路由器获取信息，计算出从主干到每个非主干区域的最短路径，这一信息再分发给区域边界路由器，由该路由器在它的区域中广播该消息。通过这个消息，路由器在转发到其他区域的分组时，可以选择到主干区域的最合适出口（区域边界路由器）。

2.2.3　虚拟专用网和网络地址转换

1．虚拟专用网

（1）专业地址（Private Address）

由于 IP 地址紧缺，一个机构能够申请到的 IP 地址往往小于本机构所拥有的主机数。实际上，出于安全等原因，一个机构内的很多主机并不需要接入外部的 Internet，它们主要是和内部的其他主机进行通信。如果一个机构内的计算机通信也采用 TCP/IP，那么主机必须配置 IP 地址。为了解决这一问题，[RFC 1918]规定了一些专用地址。这些专用地址只能用于一个机构的内部通信，而不能用于和 Internet 上的主机通信。换言之，专用地址只能用本地地址，而不能用做全球地址。在 Internet 中的所有路由器对目的地址是专用的数据报一律不进行转发。[RFC 1918]规定的专用地址如下。

1）10.0.0.0～10.255.255.255（或记为 10/8，它又称为 24bit 块）。

2）172.16.0.0～172.31.255.255（或记为 172.16/12，它又称为 20bit 块）。

3）192.168.0.0～192.168.255.255（或 192.168/16，它又称为 16bit 块）。

上面的 3 个地址块分别相当于一个 A 类网络、16 个连续的 B 类网络和 256 个连续的 C 类网络。

采用这样的专用 IP 地址的互连网络称为专用网。显然，全世界可能有很多的专用互连网络具有相同的专用 IP 地址，但这并不会引起麻烦，因为这些专用地址仅在本机构内部使用，专用 IP 地址也称为可重复地址（Reusable Address）。

（2）虚拟专用网

有时一个很大的机构有许多部门分布在相距很远的一些地点，而在每一个地点都有自己的专用网。假定这些分布在不同地点的专用网需要经常进行通信，可以有以下两种方法。

第一种方法是租用电信公司的线路为本机构专用，用来把分布在不同地点的专用网互连起来，形成企业的专用网。这种方法的好处是简单方便，但线路的租金太高。

第二种方法是利用 Internet（即公共互联网）来把分布在不同地点的专用网互连起来，实现本机构的专用网。因为是通过 Internet（而并没有用专网）来连接分散在各地的本地网络，因此这样的专用网又称为虚拟专用网（Virtual Private Network，VPN）。“虚拟”即“好像是”，但实际上不是。VPN 只是在效果上和真正的专用网一样。

2．网络地址转换

下面讨论另一种情况，就是在专用网内部的一些主机本来已经分配到了本地 IP 地址，但现在又想和 Internet 上的主机通信（并不需要加密），那么应当采取什么措施呢？

最简单的办法就是设法再申请一些全球 IP 地址，但这在很多情况下不容易做到，因为全球 IP 地址已经所剩不多。目前使用得最多的方法是采用网络地址转换（Network Address Translation，NAT）。

网络地址转换方法是在 1994 年提出的，这种方法需要在专用网连接到 Internet 的路由器上安装 NAT 软件。装有 NAT 软件的路由器称为 NAT 路由器，它至少有一个有效的外部全球地址 IP_G。这样，所有使用本地地址的主机在和外界通信时，都要在 NAT 路由器上将其本地地址换成 IP_G 才能和 Internet 连接。

例如，当内部主机 X 用本地地址 IPX 和 Internet 上的主机 Y 通信时，它所发送的数据报必须经过 NAT 路由器。NAT 路由器将数据报的源地址 IPX 转换成自己的全球地址 IP_G，但目的地址 IPY 保持不变，然后发送至 Internet。当 NAT 路由器从 Internet 收到主机 Y 发回的数据报时，知道数据报中的源地址是 IPY，而目的地址是 IP_G。根据原来的记录（这个记录称为 NAT 转换表），NAT 路由器知道这个数据报是要发送给主机 X 的，因此 NAT 路由器将目的地址 IP_G 转换为 IP_X，转发给最终的内部主机 X。

如果 NAT 路由器具有多个全球 IP 地址，那么就可以同时将多个本地地址转换为全球 IP 地址，因而可以使多个拥有本地地址的主机能够和 Internet 上的主机进行通信。

还有一种 NAT 转换表，它将利用传输层的端口号，这样就可以用一个全球 IP 地址使多个拥有本地地址的主机同时和 Internet 上的不同主机进行通信。

2.2.4　下一代网际协议 IPv6

1．IPv6 产生的背景和特点

现在使用的 IPv4 是在 20 世纪 70 年代设计的，80 年代初 TCP/IP 开始使用时，Internet 上大约只有 1000 台主机，IPv4 的地址空间可以在 1670 万个网络上接入 40 亿台主机，这在当时简直就是天文数字，如此庞大的地址数目也导致了早期的地址分配不尽合理。例如，申请到一个 B

类地址的用户，可以使用 65000 多个 IP 地址，但实际上并没有接入这么多主机，相当一部分地址被闲置并且不能再分配。另外，由于历史的原因，美国一些大学和公司占用了大量的 IP 地址。例如，MIT 和 AT&T 等就占用了 1600 多万个 IP 地址，而分配给我国的地址数不如美国一所大学，在网络技术快速发展的一些国家没有足够的 IP 地址可以使用。

Internet 规模爆炸式的增长远远超出互联网的先驱们制定 TCP/IP 时的想象，只有 C 类地址还有剩余，IPv4 将在近几年消耗殆尽，有人称此为“网络泰坦尼克号危机”。另外，Internet 规模的迅速增长也导致了路由表的极度膨胀。

为了缓解地址危机的发生，产生了相应的解决技术：无类型域间路由（CIDR）和 NAT 等，但也只是短期的补救方案。

为此，Internet 工程部 IETF 在 1992 年 6 月就提出要制定下一代的 IP，即 IPng（IP Next Generation）。IPng 称为 IPv6。1998 年 12 月的 IPv6RFC 文档[2460～2462]已成为 Internet 的草案标准，相应 IPv6 的 ICMP 的新版本是 ICMPv6（RFC 2463 草案标准）。

IPv6 和 IPv4 相比，主要的改进和特点如下。

1）大大扩充了地址空间，有多级地址结构和无类别地址。IPv6 地址增大到了 128bit，增加了地址的层数。

2）使用新的简化的首部格式。IPv6 使用一种新的数据报格式，首部 IPv4 的 13 个字段减少 8 个字段，使用了固定长度的首部和扩展首部。

3）简化了协议，加快了数据报转发的速度。例如，取消首部检验和字段，改进了分片机制，只是在源进行分片。

4）对流的支持。流是特定源和目的之间的数据报序列，IPv6 报头中有专门的流标签字段。路由器根据流标签对流中的数据报进行同样的处理，加快了数据报的处理速度。IPv6 同时定义了流的优先级，以支持不同类型的业务需求，提供 QoS 支持。

5）安全功能。IPv6 将 IP 安全（IPSec）的认证首部（Authentication Header，AH）和封装安全净荷（Encapsulation Security Payload，ESP）作为标准配置，规定了身份认证扩展首部和封装安全净荷扩展首部，以保证信息传输的安全性。

6）即插即用（Plug&Play）功能。计算机接入 Internet 时自动获取 IP 地址，端点设备可以将路由器发来的网络前缀和本身的网卡地址综合，自动生成自己的 IP 地址。对基于 IP 的第三代移动通信，即插即用更是有必要，因为当移动终端进入新的子网后必须立即获得新的地址，来不及进行手工配置。

7）扩展的地址层次结构。IPv6 由于地址空间很大，因此可以划为更多的层次。

2．IPv6 数据报

（1）IPv6 数据报的格式

IPv6 仍支持无连接的传送，但将协议数据单元（PDU）称为分组，而不是 IPv4 的数据报。为了方便起见，本书仍采用数据报这一名词。

IPv6 数据报的格式如图 2-41 所示。最前面的是基本首部（Base Header），其后有可选的 0 到多个扩展首部（Extension Header），后面是数据区。

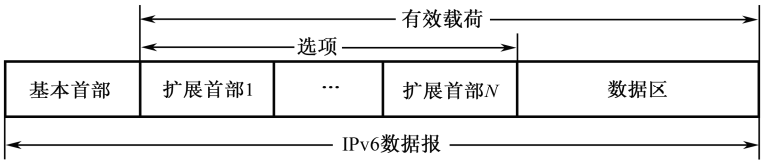
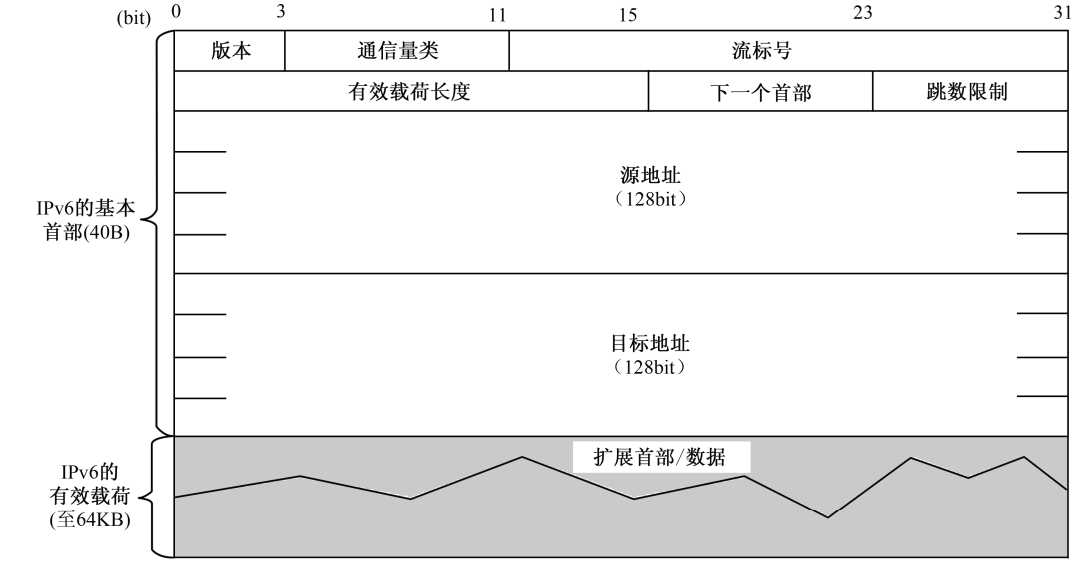


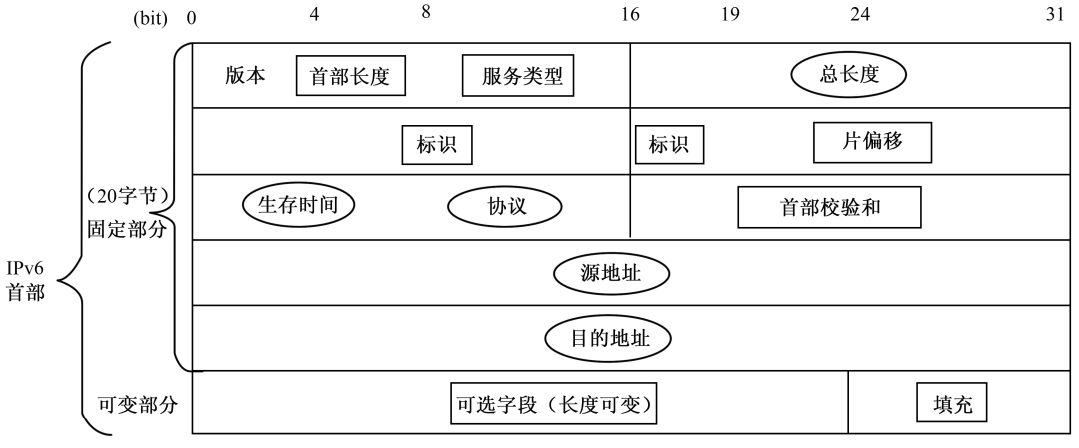
图 2-41　IPv6 数据报的格式

（2）IPv6 数据报的基本首部

IPv6 数据报的基本首部格式如图 2-42（a）所示，长度为 40 字节。IPv6 基本首部的不少字段和 IPv4 首部中的字段意义相同。



(a)IPv6数据报的基本首部格式



(b)IPv4数据报的首部格式

图 2-42　IPv6 数据报的基本首部格式和 IPv4 数据报的首部格式

IPv6 基本首部中的各个字段解释如下。

1）版本（Version），占 4bit，用来指明协议的版本。对于 IPv6，该字段总是 6。

2）通信量类（Traffic Class），占 8bit，用来区分不同的 IPv6 数据报的类别或优先级。目前正在进行不同的通信量类性能的实验。

3）流标号（Flow Label），占 20bit。IPv6 的一个新的机制是支持资源预分配，并且允许路由器将每一数据报与一个给定的资源分配相联系。IPv6 提出流（Flow）的抽象概念。所谓“流”，就是互连网络上从特定源点到特定终点（单播或多播）的一系列数据报（如实时音频或视频传输），而在这个“流”所经过的路径上的路由器都保证指明的服务质量，所有属于同一个流的数据报都具有同样的流标号。

4）有效载荷长度（Payload Length），占 16bit，用来指明 IPv6 数据报除基本首部以外的字节数（所有扩展首部都算在有效载荷之内），这个字段的最大值是 64KB。

5）下一个首部（Next Header），占 8bit。它的具体含义如下。

① 当 IPv6 数据报没有扩展首部时，下一个首部字段的作用和 IPv4 的协议字段一样，它的值指出了基本首部后面的数据应交付给 IP 上面的哪一个高层协议。

② 当出现扩展首部时，下一个首部字段的值就标志后面第一个扩展首部的类型。

6）跳数限制（Hop Limit），占 8bit，用来防止数据报在网络中无限期地存在。源站在每一个数据报发出时即设定某个跳数限制。每个路由器在转发数据报时，要先将跳数限制字段中的值减 1。当跳数限制的值为零时，就要将此数据报丢弃。

7）源地址，占 128bit，是数据报发送站的 IP 地址。

8）目的地址，占 128bit，是数据报接收站的 IP 地址。

9）从图 2-42（a）可以看出，IPv6 基本首部和 IPv4 首部相比，取消了不少功能。为了和 IPv4 数据报的首部进行对比，我们在图 2-42（b）中重新画出 IPv4 数据报的首部，将在 IPv6 数据报中被取消的字段加上方框号（共 8 个），而将有变化的字段加上椭圆框记号（共 5 个）。

3．IPv6 数据报的扩展首部

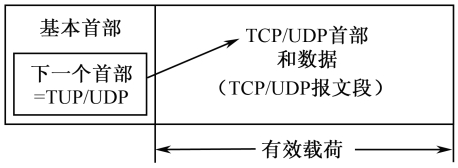
1）扩展首部及下一个首部字段。IPv6 的扩展首部与 IPv4 的选项相似，通过使用某些可选的扩选首部，指明源站希望对数据报进行的某些特殊处理，具有很大限度的灵活性。目前已定义了 6 种 IPv6 扩展首部，如表 2-6 所示。

表 2-6　IPv6 扩展首部及功能

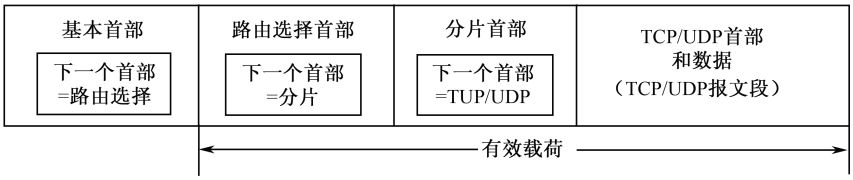
扩展首部	功　能
逐跳选项（Hop By Hop Options）	给路由器的各种信息
路由选项（Routing Options）	源站指定严格或宽松的路由
分片选项（Fragmentation Options）	数据片的分片控制
目标选项（Destination Options）	给目标的附加信息
身份认证选项（Authorication Options）	对发送主机身份的验证
净荷安全封装选项（Encapsulating Security Options）	为数据报提供加密

每一个扩展首部都由若干个字段组成，它们的长度也各不相同。但所有扩展首部的第一字段都是 8bit 的“下一个首部”子段，标识下一个扩展首部的类型。例如，如果下一个扩展首部都是逐跳选项、路由选项、分片选项或身份认证选项，则“下一个首部”字段的值分别是 0、43、44 和 51。IPv6 数据报的最后一个扩展首部“下一个首部”字段指明其后的数据区的数据类型，如 TCP=6、UDP=17 等。

图 2-43（a）表示了数据报无扩展首部时的情况。图 2-43（b）表示了在基本首部后面有两个扩展首部的情况。



（a）无扩展首部



（b）有两个扩展首部

图 2-43　IPv6 的扩展首部

2）扩展首部举例。下面以分片扩展首部为例来说明扩展首部的作用。

IPv6 将分片限制为由源站来完成。源站可以采用保证的最小 MTU（1280 字节），或者在发送数据前完成路径最大传送单元发现（Path MTU Discovery），以保证每个数据报片都小于此路径的 MTU。因此，分片是端到端的，路径途中的路由器不允许进行分片。

IPv6 基本首部不包括用于分片的字段，而是在需要分片时，源站在每一数据片的基本首部的后面插入一个小的分片扩展首部，它的格式如图 2-44 所示。

图 2-44　分片扩展首部格式

IPv6 保留了 IPv4 分片的大部分特征，其分片扩展首部共有以下几个字段。

- 下一个首部（8bit）：指明紧接着这个扩展首部的下一个首部。
- 保留（10bit）：为今后使用。该字段在第 8～15bit 和第 29～30bit。
- 片偏移（13bit）：指明本数据报片在原来的数据报中的偏移量，以 8 字节为单位表示，所以每个数据报片的长度必须是 8 字节的整数倍。

- M（1bit）：M=1 表示后面还有数据报片，M=0 则表示这已是最后一个数据报片。
- 标识符（32bit）：由源站产生的、用来唯一标识数据报的一个 32bit 数。每产生一个新的数据报，就将这个标识符加 1。采用 32bit 的标识符，可使得从源站发送到同样的目的站的数据报在生存时间内无相同的标识符（即使是高速网络）。

4．IPv6 地址

（1）IPv6 地址空间

IPv6 数据报的目的地址可以包括 3 种基本类型的地址。

1）单播（Unicast）：即传统的点对点通信。

2）多播（Multicast）：即点对多点的通信，即数据报交付到一组计算机中的每一位。IPv6 没有采用广播的术语，而是将广播看作多播的一个特例。

3）任播（Anycast）：这是 IPv6 增加的一种类型。任播地址表示了目的站是一组计算机，但发到一个任播地址的数据报只传送该组的某一个成员，通常是距离（路由协议的距离度量）最近的一个。例如，利用任播可以访问离用户最近的 DNS 服务器和文件服务器等。

在 IPv6 中，每个地址占 128bit，地址空间大于 3.4×10^{38}。如果整个地球表面（包括陆地和水面）都覆盖着计算机，那么 IPv6 允许每平方米拥有 7×10^{23} 个 IP 地址。如果地址分配速率是每微秒分配 100 万个地址，则需要 1019 年的时间才能将所有可能的地址分配完毕。可见在想像的将来，IPv6 的地址空间是不可能用完的。

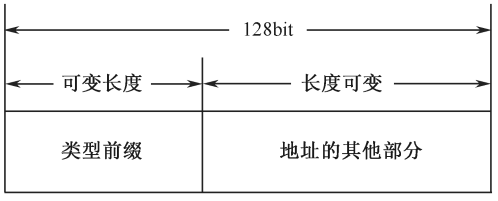


图 2-45　IPv6 的地址结构

IPv6 将 128bit 地址空间分为两大部分。第一部分是可变长度的类型前缀，它定义了地址的类型。第二部分是地址的其余部分，其长度也是可变的。图 2-45 表示了 IPv6 的地址结构，IPv6 的地址类型前缀如表 2-7 所示。

表 2-7　IPv6 的地址类型前缀

类型前缀（二进制）	地址的类型	占地址空间的份额
00000000	保留（与 IPv4 兼容）	1/256
00000001	未指派	1/256
0000001	保留给 OSI 的 NSAP 地址	1/128
0000010	保留给 Novell NetWare IPX 地址	1/128
0000011	未指派	1/128
00001	未指派	1/32
0001	未指派	1/16
001	可聚合的全球单播地址	1/8
010	未指派	1/8
011	未指派	1/8
100	基于物理的单播地址保留	1/8
101	未指派	1/8

续表

类型前缀（二进制）	地址的类型	占地址空间的份额
110	未指派	1/8
1110	未指派	1/16
11110	未指派	1/32
111110	未指派	1/64
11111100	未指派	1/128
111111100	未指派	1/512
1111111010	本地链路单播地址	1/1024
1111111011	本地地点单播地址	1/1024
11111111	多播地址	1/256

（2）IPv6 地址的记法

如何表示 IPv6 地址？如果还有 IPv4 的点分十进制记法来标记 128 位长的 IPv6 地址，显得太长，可读性和可用性太差，使用起来颇为不便。为此，IPv6 使用冒分十六进制记法（Colon Hexadecimal Notation，简写为 Colon Hex），它把每个 16bit 的值用十六进制值表示，各值之间用冒号分隔。例如，“68E6:8C64:FFFF:FFFF:0:1180:960A:FFFF”，这里将 0000 中的前 3 个 0 省略了。例如，3 个 0 后面一个 F（000F）可缩写为 F。

为进一步简化和方便使用，冒分十六进制记法还使用以下两种技术。

- 冒分十六进制记法可以允许零压缩（Zero Compression），即一连串连续的零可以被一对冒号取代。例如，“FF05:0:0:0:0:0:0:B3 可以写成如下简洁形式，即 FF05: : B3”。

IPv6 规定，在一个 IPv6 地址中只能使用一次零压缩。

- 冒分十六进制记法可以和点分十进制记法的后缀联合使用。这种结合表示方法在 IPv4 向 IPv6 的过渡阶段特别有用。例如，下面的串是一个合法的冒分十六进制记法：“0:0:0:0:0:0:128:10:2:1”。在这种记法中，虽然被冒号分隔的每个值是一个 16bit 的量，但每个点分十进制部分的值则指明了 1 字节（8bit）的值。再使用零压缩即可得出“：：128.10.2.1”。

另外，CIDR 的斜线表示法在 IPv6 地址中仍然可用。例如，60bit 的前缀 12AB00000000CD3（十六进制表示的 15 个字符，每个字符代表 4bit）可记为“12AB:0000:0000:CD30:0000:0000:0000:0000/60”、“12AB: ：CD30:0:0:0:0/60”、“12AB:0:0:CD30: : /60”。

5．从 IPv4 向 IPv6 过渡的基本方法

由于现在 Internet 使用 IPv4 协议的路由器的数量太多，因此规定一个日期实现从 IPv4 向 IPv6 的转变是不可行的。IPv4 向 IPv6 过渡只能采用逐步推进的办法，同时要求新安装的 IPv6 系统能够向后兼容，能够接收、路由选择和转发 IPv4 分组。

目前，人们研究最多的两种过渡方法是双协议栈（Dual Stack）技术和隧道（Tunneling）技术。

1）双协议栈：指在完全过渡到 IPv6 之前，使一部分主机和路由器装有两个协议，即 IPv4

协议和 IPv6 协议。因此这种主机既能够与 IPv6 的系统通信，又能够与 IPv4 的系统通信。具有双协议的主机或路由器应当具有两个 IP 地址：一个 IPv6 地址和一个 IPv4 地址。IP 主机在与 IPv6 主机通信时采用 IPv6 地址，而与 IPv4 主机通信时采用 IPv4 地址。但双协议栈主机需要通过域名系统（Domain Name System，DNS）来查询目的主机的地址类型。

2）隧道技术：指在 IPv6 分组进入 IPv4 区域时，将其封装成为 IPv4 数据报，整个 IPv6 数据分组变成了 IPv4 数据分组的数据部分；当 IPv4 数据报离开 IPv4 区域时，再将其数据部分交给主机的 IPv6 协议栈，这就好像在 IPv4 区域中打通了一个 IPv6 隧道来传输 IPv6 数据分组。

小结

本章讨论了 Internet 协议族中的数据链路层协议和网络层协议。我们描述了很多链路都具有的一个重要特性——MTU，相关的一个概念是路径 MTU。本章内容只覆盖了当今 TCP/IP 所采用的部分数据链路公共技术。TCP/IP 成功的原因之一是它几乎能在任何数据链路技术上运行。

IP 路由操作对于运行 TCP/IP 的系统来说是最基本的，不管是主机还是路由器。路由表项的内容很简单，包括 5bit 标志、目的 IP 地址（主机、网络或默认）、下一站路由器的 IP 地址（间接路由）或者本地接口的 IP 地址（直接路由）及指向本地接口的指针。主机表项比网络表项具有更高的优先级，而网络表项比默认项具有更高的优先级。系统产生的或转发的每份 IP 数据报都要搜索路由表，它可以被路由守护程序或 ICMP 重定向报文修改。在下一章，我们将讨论传输层协议的工作原理。

参考文献

[1] 谢希仁．计算机网络 5 版[M]. 北京：电子工业出版社，2008.
[2] 刘东飞，李春林．计算机网络[M]. 北京：清华大学出版社，2007.
[3] 周德新，张会兵，刘联海．计算机通信网基础[M]. 北京：机械工业出版社，2008.
[4] 高传善，毛迪林，曹袖．计算机网络教程[M]. 北京：高等教育出版社，2008.
[5] 张曾科，吉吟东．计算机网络. 3 版 [M]. 北京：清华大学出版社，2009.
[6] 金志刚．计算机网络[M]. 西安：西安电子科技大学出版社，2009.

第 3 章 传输层

传输层位于 OSI 模型的第四层，是应用层和网络层的界面与桥梁，是网络体系结构中最重要的一层。传输层的主要作用是在通信子网提供服务的基础上，为源计算机和目的计算机提供可靠、透明的数据传输。传输层的另一个重要作用是进一步加强底层网络数据传输服务质量（QoS），在不可靠的 IP 服务基础上，提高传输的可靠性。

3.1 传输层简介

3.1.1 传输层的端–端通信及与 QoS 的关系

1．点-点通信与端-端通信

由物理层、数据链路层和网络层组成的通信子网为网络环境中的主机提供点-点通信服务，通信子网只提供一台机器到另一台机器之间的通信，不会涉及程序或进程的概念。

传输层提供的是网络环境中的主机的端-端进程通信服务。端-端信道由一段段的点-点信道构成。端-端协议建立在点-点协议上，提供应用程序进程之间的通信手段。

2．传输层与 QoS 的关系

通信子网对于数据的传输是有差错的。设计传输层的目的是提高传输服务的可靠性与保证 QoS，弥补通信子网服务的不足。

3.1.2 传输层协议简介

1．网络层、传输层与应用层的逻辑关系

传输层的最终目标是向应用层的进程提供有效、可靠的服务。为了达到这一目标，传输层

利用了网络层所提供的服务，传输层中完成这一工作的硬件或者软件称为传输实体（Transport Entity）。传输实体可能在操作系统内核中，或在一个单独的用户进程内，也可能包含在网络应用程序中。网络层、传输层和应用层的逻辑关系如图 3-1 所示。

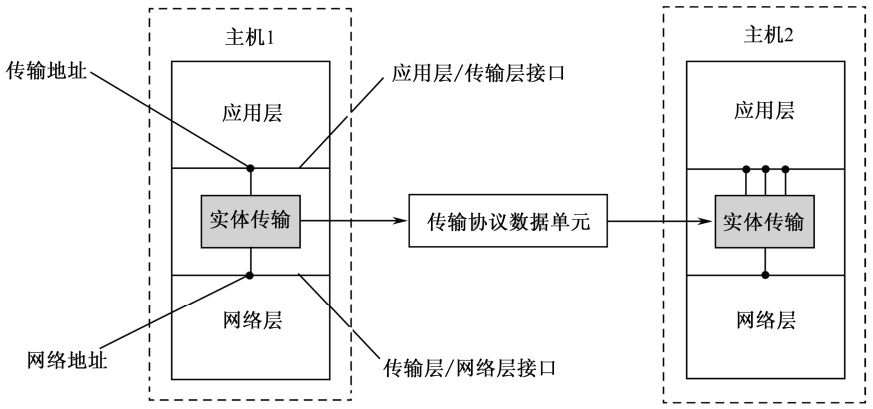


图 3-1　网络层、传输层和应用层的逻辑关系

2．面向连接和无连接服务

1）传输服务同样也有面向连接和无连接服务两种类型。

2）面向连接的传输服务连接包括 3 个阶段：建立连接、数据传输和释放连接。而无连接传输服务没有建立连接和释放连接过程。

3）网络层是通信子网的一部分，并且是由电信公司来提供的。如果网络层提供的面向连接的服务不可靠（如频繁丢失分组），而用户无法对子网加以控制，因此它们需要在网络层上再增加一层以改善服务质量。

4）传输层会对分组丢失、线路故障进行检测，并采取相应的补救措施。有了传输层后，应用层的网络应用程序不必担心不同的子网接口和不可靠的数据传输。传输层具有隔离通信子网的技术差异性及传输可靠性不足的特点。

3．传输协议数据单元

传输层之间传输的报文称为传输协议数据单元（Transport Protocol Data Unit，TPDU）。TPDU 有效载荷是应用层的数据，传输层在 TPDU 有效载荷之前加上 TPDU 头，就形成了 TPDU。

4．传输协议运行环境的特点

传输协议和数据链路层协议相似。它们都必须解决差错控制、分组拆装、流量控制等问题。但两者在协议运行环境方面差别较大，图 3-2 给出了两者的比较。

1）在数据链路层，两个路由器通过点-点链路直接通信；而在传输层，这个点-点链路被整个通信子网取代，这一差异对协议产生了很多重要的影响。在数据链路层，不必为一个路由器指明它要与哪个路由器通信，因为每条输出线路都连接着唯一的路由器。因此，传输层连接建立过程比较复杂。

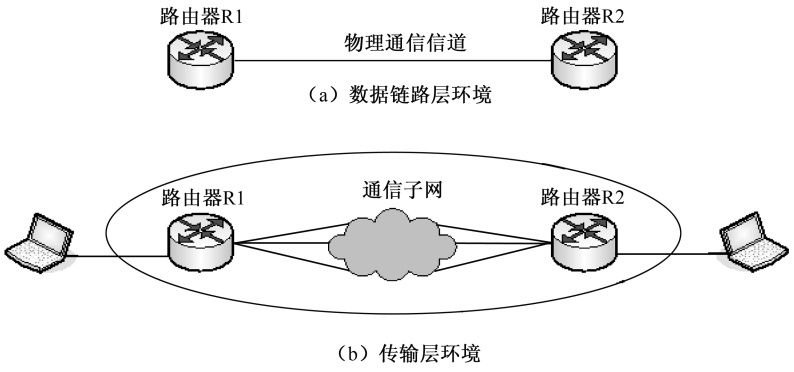


图 3-2　传输层协议和数据链路层协议运行环境的比较

2）数据链路层和传输层之间最大的区别是存储能力。当路由器发送一帧时，该帧可以直接传输到下一个路由器，但该帧经过通信子网时有可能会丢失。正常情况下，它也需要一段传输延迟才能到达目的主机，能将分组“存储”起来的能力使得传输层需要使用特殊的协议。

3）数据链路层和传输层都需要考虑数据缓冲和流量控制，但在传输层会出现大量的、动态变化的连接要求。在传输层中，由于需要管理很大数目的连接，因此必须为每一个连接分配多缓冲区。

3.1.3　传输服务与 QoS

1．传输服务

服务（Service）这个术语在计算机网络中是一个很重要的概念。“服务”是描述相邻之间关系的重要概念。

1）服务体现在网络中低层向相邻上层提供的一组操作。低层是服务的提供者，高层是服务的用户。

2）任何服务都有质量 QoS 的问题。

3）对于面向连接的传输层，衡量其 QoS 的重要指标有以下几个。

① 连接建立时延/释放时延：从传输服务用户要求建立连接到收到连接确认之间所经历的时间，它包括了远程传输实体的处理延迟，连接建立延迟越短，服务质量越好。

② 连接建立/释放失败概率：在最大连接建立延迟时间内，连接未能建立的可能性，如由于网络拥塞、缺少缓冲区或其他原因造成的失败。

③ 传输时延：指从源计算机传输用户发送报文开始，到目的计算机传输用户接收到报文为止的时间。每个方向的传输延迟是不同的。

④ 吞吐量：在某个时间间隔内测得的每秒传输用户数据的字节数。每个传输方向分别用各自的吞吐量来衡量。

⑤ 残余误码率：用于测量丢失或乱序的报文数占整个发送报文数的百分比。理论上残余误

码率应为零，实际上它可能是一较小的值。

⑥ 安全保护：为传输用户提供了传输层的保护，以防止未经授权的第三方读取或修改数据。

⑦ 优先级：为传输用户提供用以表明哪些连接更为重要的方法。当发生拥塞事件时，确保高优先级的连接先获取服务。

⑧ 恢复功能：当出现内部问题或拥塞情况时，传输层本身自发终止连接的可能性。

其中，很多指标与低层协议的 QoS 直接相关，不是本层都能单独决定的。例如，传输时延指标很大程度上就取决于通信子网本身的结构和性能，广域网总是比局域网的时延大，无论高层协议的设计如何合理，它只能尽可能减少时延的增加，而不可能减少时延。

网络中的 *N* 层总是要向 *N*+1 层提供比 *N*−1 层更完善、更高质量的服务，否则 *N* 层就没有存在的价值。这个思想贯穿于在整个网络层次结构和网络设计中，传输层的协议设计也是遵循这一基本的设计思想。

2．QoS

在计算机网络中，通信子网是指网络层及其以下的部分。通信子网概念的形成除技术因素外，还有一定的历史原因，通信子网的概念是在广域网结构的研究中提出来的。通信子网出现之初，许多广域网用租用点-点通信线路来连接它们的接口报文处理机 IMP，后来的一些广域网直接建立在提供分组传输服务的公用数据网（如 X.25 网等）上。在传统的业务分工中，数据通信属于电信公司的业务范围。电信公司所提供的数据传输服务，要比用户建设自己的专用通信子网经济得多。因此，通信子网往往是计算机网络组建部门和用户无法控制的部分。

通信子网到底提供什么服务及什么样的质量服务（QoS），这些问题是组建部门和用户必须面对的。如果通信子网服务能够满足用户需求，那么传输层就可能变得很简单。如果通信子网服务不能满足用户需求，那么传输层必须对通信子网服务加以完善，这样传输层协议就可能变得比较复杂。因此传输层在网络分层结构中起着承上启下的作用，它将向高层屏蔽通信子网的技术、设计的差异与服务质量的不足，向高层提供一个标准的、完善的通信服务。由于传输层的特殊地位，人们一般将网络层模型分为两部分：传输层及其以下各层称为传输服务提供者，传输层以上各层称为传输服务用户。

在传输层设计中，有两种可能的方法。第一种方法是针对每一种通信子网和所需要的传输服务都设计一个传输协议。这种方式的好处在于可以有的放矢地解决问题，没有额外开销，效率很高，但传输协议缺乏通用性。第二种方法是针对通信子网可能的服务类型和各种传输服务需求，设计一个通用的传输层协议。这种标准化的设计思想可能把传输层协议变得大而全，但效率低。一种折中的方案是将通信子网分类，针对每一类通信子网设计相应的传输协议，既保证效率，又不失通用性。

通信子网分类的标准是数据传输的可靠性。在子网的服务类型和 QoS 指标中，有很多指标是由底层物理网络技术决定的，传输层可以加以改善的是它的可靠性，包括是否有分组丢失、重复和乱序等。在 QoS 的指标中，时延等是反映通信子网物理特性的指标，是无法通过传输层协议来改善的，而可以通过传输层协议改善的指标是连接建立/释放失败概率、残余误码率等可

靠性。

人们往往以可靠性作为通信子网标准，并将通信子网分为 A、B、C 这 3 类。其中，A 类通信子网的网络层协议是面向连接的，可靠性最好，但只有少数的局域网可以提供近似于 A 类通信子网的服务。B 类也是面向连接的，单个的数据报传输是可靠的，但网络连接不可靠，会因为网络拥塞、系统崩溃等而中断。无连接服务的广域网、分组无线交换网等无连接通信子网属于 C 类通信子网，C 类通信子网的可靠性最差。

按照通信子网面向连接与无连接的特点，依据通信子网的可靠性，传输层协议也有两类。例如，在 TCP/IP 体系中，虽然通信子网的网络层都采用无连接的 IP，但根据物理技术的不同，传输层的 UDP 和 TCP 有不同的适用范围，UDP 适用于可靠性较高的子网，TCP 适用于可靠性较差的广域网。

3.2　UDP

UDP 是一个简单的面向数据报的传输层协议，主要用来支持那些需要在计算机之间传输的网络应用。它包括网络视频会议系统在内的众多客户/服务器（Client/Server，C/S）模式的网络应用都需要使用 UDP。UDP 是对 IP 协议簇的扩充，它增加了一种机制，发送方使用这种机制可以区分一台计算机上的多个接收者。每个 UDP 报文除了包含某用户进程发送的数据外，还有报文目的端口的编号和报文源端口的编号，从而使 UDP 得到扩充，在这两个用户进程之间的递送数据报成为可能。

UDP 是依靠 IP 来传输报文的，因而它的服务和 IP 一样是不可靠的。这种服务不用确认，不对报文排序，也不进行流量控制，UDP 报文可能会出现丢失、重复、失序的现象。进程的每个输出操作都正好产生一个 UDP 层的数据报，并组装成一个待发送的 IP 数据报。UDP 不提供可靠性，应用程序传给 IP 层的数据发送出去，但是并不保证它们能到达目的地。应用程序必须关心 IP 数据报的长度，如果它超过网络的 MTU，那么就要对 IP 数据报进行分片。

3.2.1　UDP 报文的格式

每一个 UDP 报文成为一个用户数据报（User Datagram）。从概念上讲，用户数据报分为两个部分，即 UDP 首部和 UDP 数据区，如图 3-3 所示。

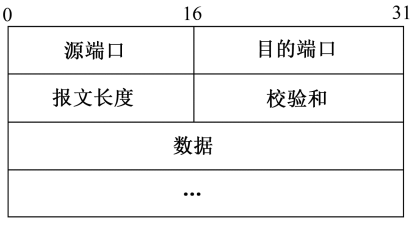


图 3-3　UDP 报文的格式

1．UDP 首部

首部被分为 4 个 16bit 的字段，分别是源端口（Source Port）、目的端口（Destination Port）、长度和校验和（Checksum）。源端口字段和目的端口字段包含了 16bit 的 UDP 端口号，以便在各个等待接收报文

的进程之间对数据报进行多路分解操作。其中源端口字段是可选的，若选用，则指定应答报文应该发往目的端口；若不选用，其值应该为 0。报文长度（Length）字段记录了该 UDP 数据报中的 8 位组数，这个长度包括了 UDP 首部和用户数据，因此，报文长度字段的最小值是 8，即首部的长度。

UDP 的校验和字段是可选的，如果该字段值为 0，就说明不进行校验。设计中把这个字段作为选项的目的，是为了那些在高可靠性的局域网上使用 UDP 的实现者能够尽量减少开销。但是，IP 对 IP 数据报中的数据部分并不计算校验和，所以 UDP 的校验和字段提供了唯一的途径来保证数据原封不动地到达，因此使用这个字段是很有必要的。如果 UDP 报文的校验和计算出来的结果为 0，UDP 使用和 IP 同样的校验和计算方法，即将数据划分为 16bit 的单元，对每个单元计算其二进制反码，再对这些值的和求二进制反码。0 的二进制反码可以有两种表示方法，即全 0 或全 1，所以校验和为 0 并不会引起问题。当计算出的校验和为 0 时，UDP 就使用全 1 来表示这种情况。

2．UDP 的伪首部

UDP 校验和覆盖的内容超出了 UDP 数据本身的范围。为了计算校验和，UDP 把伪首部（Pseudo-header）引入数据报中，在伪首部中有一个值为 0 的填充 8 位组，用于保证整个数据报长度为 16bit 的整数倍，这样才容易计算校验和，填充 8 位组和伪首部并不随着 UDP 数据报一起传输，也不计算在数据报长度之内。为了计算校验和，我们先把 0 赋予校验和字段，然后对整个对象（包括伪首部、UDP 的首部和用户数据）计算一个 16bit 的二进制反码和。

使用伪首部的目的是检验 UDP 数据报已到达正确的目的地，理解伪首部的关键在于认识到，正确的目的地包括了特定的主机和机器上特定的协议端口，UDP 报文的首部仅仅指定了使用的协议端口号。因此为了确保报文能够正确到达目的地，发送 UDP 数据报的机器在计算校验和时把目的主机的 IP 地址和应有的数据都包括在内。在最终的接收端，UDP 软件对检验和进行检验时，要用到携带 UDP 报文的 IP 数据报首部中的 IP 地址。如果校验和正确，那么说明 UDP 数据报到达了正确主机上的正确端口。

在 UDP 检验和的计算过程中用到了伪首部长度为 12 的 8 位组，其结构如图 3-4 所示。

图 3-4　UDP 伪报头格式

伪首部的源 IP 地址字段和目的 IP 地址字段记录了发送 UDP 报文时使用的源 IP 地址和目的 IP 地址。协议字段指明了所使用的协议类型代码（UDP 是 17），而 UDP 长度字段是 UDP 数据报的长度（伪首部的长度不计算在内）。接收方进行正确性验证的时候，必须要把这些字段的信息从 IP 报文的首部抽取出来，以伪首部的格式进行装配，然后重新计算校验和。

3.2.2　UDP 的封装与协议的分层

在第 2 章介绍的网络层协议和网际协议 IP 的关系结构中，UDP 位于 IP 层之上。应用程序

访问 UDP 层，然后使用 IP 层传输数据报，如图 3-5 所示。

将 UDP 层放在 IP 层之上，表示一个 UDP 报文在 Internet 中传输时要封装到 IP 数据报中。最后，网络接口层将数据包封装到一个帧中再进行物理传输通道上的传输。封装过程如图 3-6 所示。由图 3-6 可知，IP 层的报头指明了源主机和目的主机的地址，而 UDP 层的报头指明了主机上的源端口和目的端口。IP 层和 UDP 层之间的职责是清楚而明确的：IP 层只负责在 Internet 上的一对主机之间进行数据传输，而 UDP 只负责对一台主机上复用的多个源端口或目的端口进行区分。

图 3-5　分层模型的 UDP

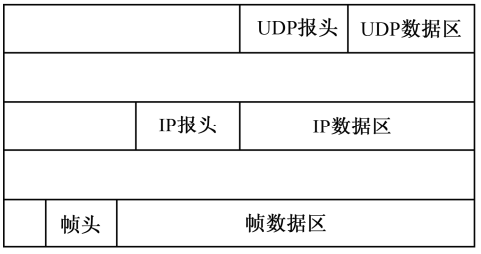


图 3-6　UDP 的封装过程

3.2.3　UDP 的复用、分解与端口

UDP 也提供复用和分解功能。它接收多个应用程序送来的数据报，把它们送给 IP 层传输，同时它接收 IP 层送来的 UDP 数据报，把它们送给对应的应用程序。

从概念上讲，所有的 UDP 软件与应用程序之间的复用和分解都要通过端口机制来实现。实际上每个应用程序在发送数据报之前与操作系统进行协商以获得协议端口和相应的端口号。凡是利用指定的端口发送数据报的应用程序都要把端口号放入 UDP 报文中的源端口字段中。

UDP 的分解操作如图 3-7 所示，UDP 从 IP 层接收了数据报之后，根据 UDP 的目的端口号分解操作。

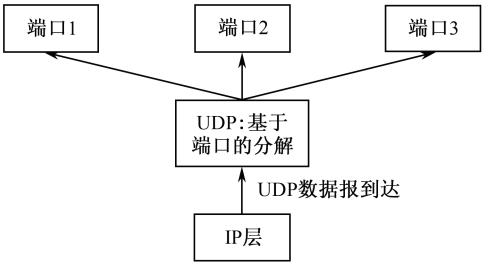


图 3-7　UDP 的分解操作

3.2.4　UDP 的可靠性

UDP 建立在 IP 之上，同 IP 一样提供无连接数据报传输。相对于 IP，它唯一增加的能力是提供协议端口，以保证进程通信。

许多基于 UDP 的应用程序在高可靠性、低延迟的局域网上运行得很好，而一旦到了通信子网服务质量（QoS）很低的互联网环境下，可能根本不能运行。原因就在于 UDP 不可靠，而这些程序本身又没有做可靠性处理。因此，基于 UDP 的应用程序在不可靠子网上必须自己解决可

靠性（如报文丢失、重复、失序和流控等）问题。

既然 UDP 如此不可靠，为何 TCP/IP 还要采纳它？最主要的原因在于 UDP 的高效率。在实践中，UDP 往往面向交易型应用，一次交易一般只有一次往返报文交换，假如为此而建立连接和撤销连接，开销是相当大的。这种情况下使用 UDP 很有效，即使因报文损失而重传一次，其开销也比面向连接的传输小。

3.3 TCP

TCP 是传输层的最重要协议，它是一个完整的传输协议的典范。除提供和 UDP 一样的进程通信能力外，其主要特点是可靠性很高。TCP 建立在不可靠的 IP 之上，TCP 的可靠性完全由自己实现，主要措施有面向连接的传输机制、超时重传机制、可变滑动窗口流量控制和拥塞控制。

3.3.1 TCP 简介

1．TCP 的功能

TCP 定义了两台计算机之间进行可靠数据传输所交换的数据和确认信息的格式，以及计算机为了确保数据的正确到达而采取的措施。

TCP 支持计算机上的多个应用程序同时进行通信，也能对收到的数据进行分解，分别送到多个应用程序。为了标识一台计算机上的多个进程，TCP 使用了协议端口（Port），每个端口被赋予一个整数以便识别。

TCP 是 TCP/IP 体系中的传输层协议，是面向连接的，可以提供可靠的、按序传送数据的服务。TCP 提供的连接是双向的，即全双工的。TCP 用一对端点来标识连接，端点由主机的 IP 地址和该主机上的 TCP 端口号组成。由于 TCP 使用两个端点来标识连接，一台机器上的某个 TCP 端口号可以被多个连接共享。

TCP 是面向数据流的协议，应用层的报文传送到传输层，它将传来的报文看作字节序列，并将字节序列划分为若干段，加上 TCP 首部，就构成了 TCP 数据传送单元，称为报文段（Segment），用 OSI 的记法就是 TCP PDU。在发送时，TCP 报文段作为 IP 数据报的数据，再加上首部后，成为 IP 数据报。接收时，IP 层将 IP 数据报首部去除后上交传输层，恢复 TCP 报文段，再去除其首部，得到应用层所需要的报文。

IP 并不保证报文的正确传输，到达的数据报也可能是乱序的。由于 TCP 进行超时、重发控制，TCP 按正确的顺序重新将这些数据报组装成报文。

2．端口的概念

UDP 和 TCP 都使用与应用层接口处的端口与上层应用进程进行的通信。应用层的各种进程要通过相应的端口才能与传输层进行交互。

在传输层与应用层的接口上所设置的端口是一个 16 位的地址，并用端口号进行标识。端口

号分为两类。一类专门分配给一些最常用的应用程序，即通用端口（Well-know Port），数值为 0～255。“通用”表示这些端口号是 TCP/IP 体系确定并公布的，因而是所有用户进程都熟知的。应用层中的各种服务器进程不断地检测分配给它们的通用端口，以便发现是否某个客户进程要与它通信。另一类则是一般的端口，用来随时分配给请求通信的客户进程。图 3-8 举出了几个常用的熟知端口。

图 3-9 说明了端口的作用。设主机 A 要使用 SMTP 与主机 C 通信。SMTP 使用面向连接的 TCP。为了找到目的主机中的 SMTP，主机 A 与主机 C 建立连接时，要使用目的主机中的熟知端口号为 25。主机 A 也要给自己的进程分配一个端口号，设分配的源端口号为 500。这就是主机 A 和主机 C 建立的第一个连接。图中的连接画成虚线，表示这种连接不是物理连接，只是个逻辑连接。

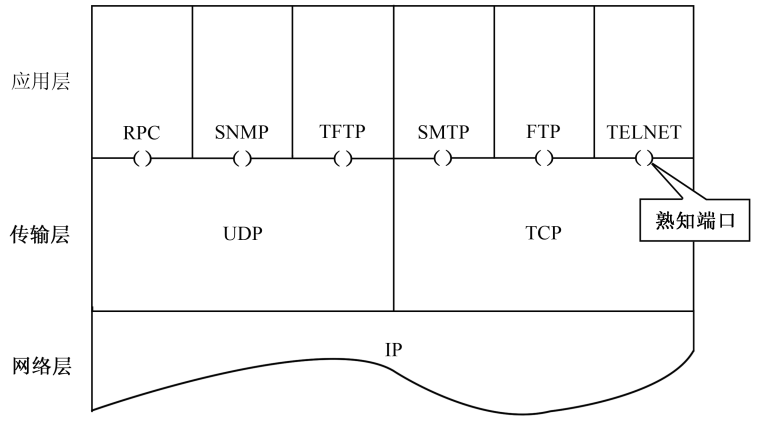


图 3-8　常用的熟知端口

现在主机 A 中的另一个进程也要和主机 C 的 SMTP 建立连接，目的端口号也为 25，但其源端口号不能与上一个连接重复。设主机 A 分配的这个端口号为 501，这是主机 A 和主机 C 建立的第二个连接。

设主机 B 也要和主机 C 的 SMTP 建立连接。主机 B 选择源端口号为 500，目的端口号当然还是 25。这是和主机 C 建立的第三个连接。这里的端口号与第一个连接的端口号相同，这属于巧合，各主机都独立地分配自己的端口号。

为了在通信时不致发生混乱，就必须把端口号和主机的 IP 地址结合在一起使用。图 3-9 中，主机 A 和 B 虽然都使用了相同的端口号 500，但只要查找 IP 地址即可知道是哪一台主机的数据。

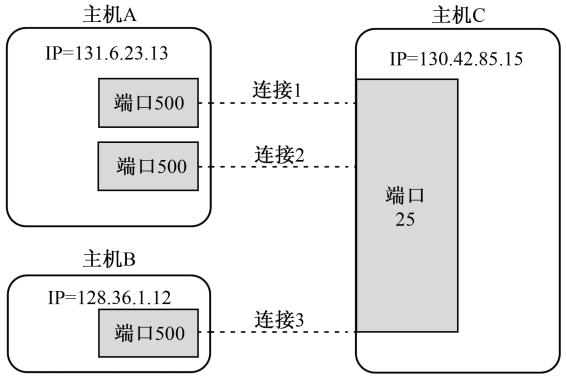


图 3-9　端口的作用

一个 TCP 连接由它的两个端点来标识，这样的端点称为插口（Socket）或套接字。插口包括了 IP 地址（32bit）和端口号（16bit），在 Internet 中，传输层通信的一对插口必须是唯一的。例如，图 3-9 中的连接 1 的一对插口是 131.6.23.13.500 和 130.42.85.15.25；而连接 2 的一对插口是 131.6.23.13.501 和 130.42.85.15.25。

上面的例子是使用面向连接的协议 TCP。若使用无连接的协议 UDP，虽然在相互通信的两个进程之间没有一条逻辑连接，但每一个方向一定有发送端口和接收端口，因而也同样可以使用端口的概念，这样才能区分同时通信的多个主机中的多个进程。

3.3.2 TCP 的编号与确认

1．数据流、报文段和编号

TCP 提供的是面向连接的可靠的流传输，这里所谓的流即数据流，指的是字节的无结构的序列。为了便于每次传输，又把数据流划分为若干段，称为报文段，每个报文段作为 TCP 的协议数据单元，PDU 被封装到一个 IP 数据报中在网上传输。报文段到达目的站后，TCP 再将它们组装为原来的数据流。

TCP 对数据流按字节编上序号，而不是按报文段编序号，TCP 将传输的报文段所携带数据的第 1 字节的序号放在报文段首部的序号字段中。序号的空间应该足够大，在 TCP 报文段格式中规定为 32bit，以便使序号循环一周的时间足够长，不至于在短时间内产生相同的序号。

TCP 定义了最大报文段生存时间（Maximum Segment Lifetime，MSL），RFC 793 规定为 120s，后来不同的 TCP 实现中也有不同的值，如 60s 等。因为 TCP 报文段是由 IP 数据报携带传送的，因此 MSL 应该大于 IP 数据报中在网上的最大生存时间（TTL）的值（120s）。在 MSL 时间内，同时出现相同序号的报文段会给接收 TCP 造成混淆，也就是说序号循环一周的时间应该大于 MSL。

产生相同的序号有以下两种情况。

一种情况是不同的 TCP 连接出现相同的初始序号（Initial Sequence Number, ISN）。ISN 不应该每次连接都从 0（或 1）开始，否则会导致不同的 TCP 连接出现重复的 ISN。如果一个 TCP 连接使用序号 0 刚刚发送了少量报文段，在小于 MSL 的时间内就崩溃了，之后它又迅速恢复并建立了新的连接，再次从序号 0 开始发送，在网上就会出现重复的序号。TCP 使用基于计数器的 ISN 方案，规定 ISN 每 4ms 加 1，可以看做一个 32bit 的计数器，循环一周的时间远远大于 120s，使不同 TCP 连接的 ISN 在 120s 内不可能相同。ISN 是在双方建立连接的过程中商定的，双方的 ISN 各自选取，使用当时计数器的值。

另一种情况是同一 TCP 连接中出现相同序号。同一 TCP 连接中产生相同的序号，与数据发送的速度直接相关。32bit 的序号空间，序号循环一周要发送 2^{32}B，在 T3 线路（45Mb/s）上需要（$2^{32}\times8$）÷（45×10^{6}）≈764s。但线路速度提高，循环一周的时间就减小，在 OC-12 线路（622Mb/s）仅需要 55s。可见，序号空间不变时，数据传输速度越高，序号循环一周的时间就越小，就可能小于 TCP 规定的 MSL。为此，可使用 TCP 的时间戳选项，发送方 TCP 在每个发送的报文段首部插入 32b 的时间戳，接收方将收到的时间戳也插入到 ACK 报文段中作为

确认。这样，32b 的时间戳和 32b 的序号组合在一起就可以解决序号循环产生的问题。

2．TCP 确认机制

TCP 采用累计确认（Cumulative Acknowledgement）方式，接收方确认已正确收到的、累计的连续数据流。TCP 使用数据流的字节序号进行确认，在确认报文段首部的确认序号字段中，收方写入的确认序号是正确收到的字节序列的最高序号加 1，表明它前面的数据流已正确收到，指明所期望接收的下一个报文段的起始序号。例如，一个 TCP 连接上发送的 ISN 为 x，发送方发送了 1500 字节长的报文段，收方的确认序号是 1500+x。

为了提高传输效率，TCP 的实现可以使用延迟确认算法（Delayed ACK Algorithm）[RFC 2581]，TCP 不必每收到一个报文段后就立即发回确认，可以推迟一段时间，在收到 1 个以上的连续报文段之后再发回确认。但延迟确认的延迟时间不能超过 500ms，太长的确认延时可能导致发送方不必要的超时重传。若在延迟等待期间接收方也有了发给发送方的数据，接收方 TCP 便将确认和自己的数据一起发出，这称为数据捎带确认（Piggybacking ACK），这也减少了一次传输过程。

若 TCP 收到了失序（Out of Order）的报文段，就立即发出一个对期望接收序号的确认，以便通知发送方可能出现了报文段丢失。对于失序但没有差错的报文段，接收方应该如何处理？TCP 标准对此没做明确规定，有待在 TCP 的实现中去解决。一种方法是将它丢弃；另一种方法是先将它暂存于接收缓冲区中，若又接收到一个报文段全部填满了接收数据中的空隙，则立即发出一个累计确认，若部分填充满接收数据中的空隙，则立即发出一个对期望接收序号的确认。后面这些策略相当于选择重传 ARQ，与简单丢弃失序报文段的做法（相当于回退-N ARQ）相比，可以减少重传，提高传输效率。

TCP 后来又引入了负确认 NAK 选项[RFC1106]，允许收方发送对某一序号的否定确认 NAK，指定发送方重发，收到之后再确认所有缓存的数据。

3.3.3　TCP 报文段

1．TCP 报文段的格式

TCP 软件之间传输的协议数据单元 PDU 为报文段，通过报文段的交互建立连接、传输数据、发出确认、通告窗口大小及关闭连接。图 3-10 给出了 TCP 报文段的格式。

一个 TCP 报文段分为首部和数据两部分。TCP 报文段首部的前 20B 是固定的，后面可以有选项，选项的字节长度应为 4 的倍数，因此，TCP 首部的最小长度是 20B。首部固定部分各字段的意义如下。

1）源端口和目的端口，各占 2B。端口是传输层与应用层的服务接口，和 UDP 一样，端口用来将多个应用程序与 TCP 进行复用和解复用。

2）序号，即发送序号，占 4B，它是本报文段所携带数据的第一字节的序号。在 TCP 传送的数据流中，每一字节都编有一个序号。由于序号字段有 32bit，可对 4GB 的数据进行编号。

3）确认序号，占 4B，用于接收方对发送方发出的数据的累计确认，说明该序号之前的数

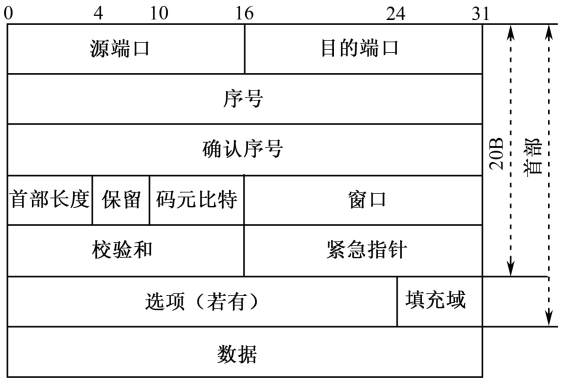


图 3-10　TCP 报文段的格式

据均已正确收到，也就是接收方期望接收的下一个报文段第一字节的序号。

4）首部长度，占 4bit，指示 TCP 报文段首部的长度，因为首部中的选项字段长度是不确定的，因此首部长度不固定。首部长度的单位是 32bit（4B）字长的字。该字段后面还留有 6bit 是保留字段，供今后使用，目前应置为 0。

5）码元比特，占 6bit，这 6bit 说明报文段的各种性质。各比特的意义规定如下。

① 紧急比特 URG：URG 比特置 1，使 16bit 的紧急指针（Urgent Pointer）有效，通知接收方，在紧急指针指示的位置之前，数据流中有紧急数据传输。如果要传输需紧急处理的数据，应尽快传送和处理。例如，TCP 用于远程登录会话服务，远程主机上的程序突然出现错误时，用户可能要从本地键盘上输入几个控制字符中断远程运行的程序，这种信号应该尽快处理。TCP 将停止在发送缓冲区积累数据，尽快将已有的数据发送出去。TCP 发送的下一个报文段将把 URG 置 1，并使首部的紧急指针字段有效。紧急指针字段的值指示紧急数据的末字节相对于本报文段序号的偏移量，两者相加可以得到紧急数据在数据流中最末字节的位置（但不知道开始位置，由应用程序处理）。即使此时接收方通告了 0 窗口，中止了连接上的数据流，发送方仍可以发送紧急报文段。接收方收到 URG 比特置 1 的报文段，使接收方应用程序进入紧急模式，以尽快处理紧急数据。接收方应用程序处理的数据超过紧急指针指示的位置，便恢复到正常操作状态。

② 确认比特 ACK：用于 TCP 确认。当 ACK=1 时，确认序号字段有效；当 ACK=0 时，确认序号无意义。

③ 急迫比特 PSH：早期的应用程序使用 PUSH 操作强迫 TCP 立即发送缓冲区积累的数据，PUSH 操作使急迫标志 PSH=1，通知收方 TCP 也立即将数据交给应用程序，不用等待后续数据。

④ 复位比特 RST：当 RST=1 时，表明出现严重差错（如主机崩溃等），必须释放连接，然后再重新建立。

⑤ 同步比特 SYN：TCP 用同步比特建立连接。当报文段的 SYN=1 和 ACK=0 时，表明它是一个连接请求；若对方同意建立连接，应在相应的报文段中使 SYN=1 和 ACK=1。

⑥ 终止比特 FIN：TCP 用终止比特来释放，当 FIN=1 时，表明欲发送的数据已经发送，要求释放 TCP 连接。

6）窗口，即通告窗口（Advertisement Window），占 2B。通告窗口告诉对方：在收到收方的下一次确认之前，发送方能够发送的数据长度不能超过此窗口的值。通告窗口实际上反映了接收方目前可用的接收缓冲区的大小，发送方可据此调节自己的发送窗口。通告窗口的单位为字节。

7）校验和，占 2B。TCP 差错检验的范围包括首部和数据两部分。但和用户数据报 UDP

一样，在计算校验和时，要在 TCP 报文段的前面加上一个 12B 的伪报头。伪报头的格式与 UDP 用户数据报的伪报头一样。对于 TCP，伪报头的协议字段的值为 6。校验和的计算方法与 UDP 相同。TCP 将丢弃校验和有差错的报文段，并做重传处理。

8）选项，长度可变。原来 TCP 只规定了一种选项，即最大报文段长度选项（Maximum Segment Size，MSS），后来为了改进 TCP 的性能，又提出了窗口比例因子选项（Window Scale Option）、负确认选项（Negative Acknowledgement，NAK）和时间戳选项（Timestamp Option）等。

2．TCP 报文段的选项

各选项的格式如图 3-11 所示，其中各字段括号内的数字是该字段的字节数。

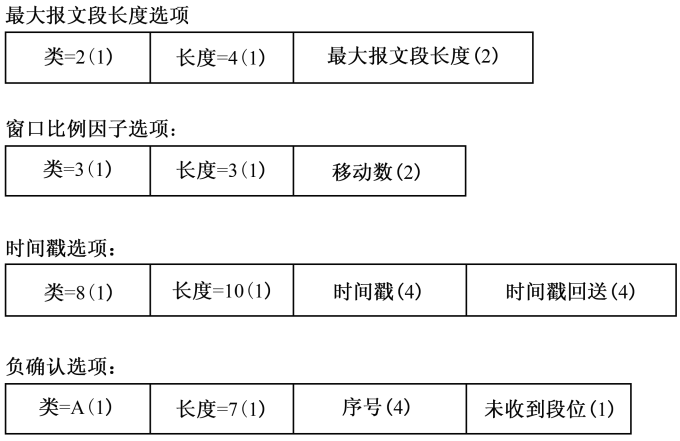


图 3-11　TCP 的选项格式

（1）MSS

MSS 指的是 TCP 报文段所携带数据的最大长度，单位为字节，不能超过此限制。一个 TCP 连接的两端必须协商一个 MSS 值。在建立 TCP 连接时，双方的 TCP 软件使用选项字段协商 MSS，4B 的选项字段的格式如图 3-11 所示。然而在互联网环境中，选择合适的 MSS 是很困难的，取值过大或过小都有问题，可能会造成网络的性能变坏。

TCP 报文段是封装在 IP 数据报中传输的，IP 数据报又是封装在物理网络的帧中传输的，每个报文段除了数据之外，还要加上至少 40B 的 TCP 和 IP 首部。极端情况下，如果 TCP 仅传输 1B 的数据，而总的 IP 数据报却有 41B，则对底层网络带宽的利用率不超过 1/41。也就是说，选择过小的 MSS 值会降低网络利用率，但 MSS 的值也不能取得过大。长的报文段构成了长的 IP 数据报，长的数据报在 MTU 较小的物理网络上传输时，IP 不得不把它分片。所有的分片必须全部正确地到达目的站才能进行重组和确认，只要有一片丢失或出错，整个数据报就要重传。因此 TCP 报文段长度越大，分片就会越多，丢失或出错的可能就越大，因而就增加了数据报重传的概率，降低了网络的性能。

从理论上讲，TCP 报文段的最佳长度 MSS 可以这样得到：在 IP 数据报从源站到目的站的路径上不被分片的前提下，它所封装的 TCP 报文段携带尽量长的数据，这个长度的最大值就是

MSS。但实际上问题并不这么简单，这是因为互联网上的路由是动态的，随着网络拓扑和网络负载等因素的变化，IP 数据报的传输路径可能改变，MSS 也随之变化。此外，最佳长度 MSS 还取决于低层协议数据单元的长度，它们也不是固定的。例如，IP 数据报有选项时，MSS 的长度就要减小。

TCP 选择最佳报文段长度的简单做法，是取建立连接时双方声明的 MSS 的较小者。如果一方没有声明 MSS，MSS 取默认值 536B。那么，所有 Internet 上的主机都能接收的 TCP 报文段的长度为 536B+20B=556B。

（2）窗口比例因子选项

计算机网络技术的飞速发展，使得当年 TCP 的某些规定不能适应今天的需要，TCP 仅 16bit 的通告窗口字段就限制了连接上的传输带宽。为此，TCP 规定了窗口比例因子，扩大通告窗口的数值范围。双方在建立连接时商定窗口比例因子。

16bit 的窗口字段只能通告 64K（65536）B 的发送窗口值，发送方至少要经过一个往返事件 RTT（Round Trip Time）才能发出发送窗口中的数据并收到确认，之后才能滑动发送窗口并继续发送。因此，TCP 在 RTT 时间内最多只能发送 64KB 的数据。TCP 连接上的最大容量为传输速率（b/s）与 RTT（s）的乘积，称为往返时延宽积，显然，它的值不能超过通告窗口。随着技术的发展和网络带宽的提高，64KB 的通告窗口已不够用。

RTT 在不同情况下是一个变化的值，一个局域网 RTT 只有几毫秒，而 Internet 上的长距离可能要几秒，一条穿越美国的 T1 线路 RTT 有 60ms 左右。当初 TCP 面对的是 56kb/s 的线路，假定 RTT 为 100ms，那么往返时延带宽积只有（$56\times10^3\times0.1$）b/8=700B。对于 1.544Mb/s 的 T1 线路，那么往返时延带宽积为 19.13KB，它们都小于当初 64KB 的窗口值。现在网络带宽大大提高。例如，对于带宽为 622Mb/s 的 OC-12 线路，在 100msRTT 的假定下，其往返时延带宽积则有 118.6×64KB；对于 OC-48 线路，其往返时延积则高达 474.5×64KB。这样，当初设计的 64KB 的窗口远不够用，所以 TCP 对 16bit 的窗口进行了扩展。

TCP 用窗口比例因子选项扩展窗口值，其格式如图 3-11 所示。窗口比例因子表示原来 16 位的窗口值向左移动的次数，每移 1 次，窗口值翻一番。窗口比例因子的最大值为 14，窗口最大可扩大 2^{14}=16384 倍，所以扩展后的窗口可达 2^{30}B=16384×64KB。

（3）NAK

TCP 采用累计确认进行差错控制，累计确认是一种正确认机制。TCP 又引入了 NAK，进行选择重传 ARQ，得到了广泛应用。NAK 的格式如图 3-11 所示。

当接收方收到了失序的报文段时，TCP 将失序的报文段缓存，并给发送方发送一个 NAK 确认，指明空缺数据的首字节序号及报文段的数目，请求发送方发送。其中报文段数目是根据最大报文段长度 MSS 计算出来的。若收方收到了有序但校验错误的报文段，并且后面又收到了正确的报文段，TCP 将丢弃错误的报文段，并将后面正确的报文段缓存，这意味着对于失序的报文段，要进行同样的处理。

发送方在收到 NAK 之后，根据收方在 NAK 中指明的空缺数据的首字节序号及报文段数目等信息，重发报文段。收方收到后，连同原来缓存的失序报文段的数据一起向发送方发回累计确认。

（4）时间戳选项

由 3.3.2 节可知，在高速线路上的 TCP32bit 的序号字段，因序号循环加快可能引起报文段重号的问题。TCP 定义了时间戳选项，可以解决这一问题。时间戳选项的格式如图 3-11 所示。

发送方 TCP 在每个发送的报文段首部插入 32bit 的时间戳，收方将收到的时间戳也插入到 ACK 报文段中作为回应。将 32bit 的时间戳和 32bit 的序号组合在一起使用，就可以解决序号循环产生的重号问题。另外，时间戳还能用于报文段往返时间 RTT 的计算。

TCP 标准推荐，每 1ms～1s 的时间间隔将时间戳的值加 1。

3.3.4　TCP 连接管理

1．建立 TCP 连接

TCP 是面向连接的协议，传输之前两端点之间要建立连接，传输结束则关闭这一连接。TCP 连接有如下特点。

1）TCP 连接是两端点之间点对点的连接，不支持一点对多点的传输和广播。

2）TCP 是全双工连接，支持双向传输，允许端点在任何时间发送数据，TCP 能够在两个方向上缓冲输入和输出的数据。

3）TCP 连接采用客户-服务器模式，主动发起连接请求的进程为客户，被动等待连接建立的进程服务器。

4）TCP 连接的端点用 IP 地址和端口号二元组来标识，而一个连接则由本地和远程的一对端点，即一个四元组（本地 IP 地址、本地端口号、远程 IP 地址、远程端口号）来标识，它是唯一的。

TCP 使用了 3 次握手（Three-way Handshake）的方式建立连接。3 次握手的过程如图 3-12 所示，主机 A 的端口 1 和主机 B 的端口 2 建立连接，共交换了 3 次报文段。

① 主机 A 发起握手，目的端点为主机 B 的端口 2。

- 生成一个随机数作为它的初始发送序号 x。
- 发出一个同步报文段，SYN=1，发送序号 seq=x，ACK=0。

② 主机 B 监听到端口 2 上有连接请求，主机 B 响应，并继续同步过程。

- 生成一个随机数作为它的初始发送序号 seq=y。
- 发出同步报文段并对主机 A 端口 1 的连接请求进行确认，SYN=1，发送序号 seq=y，ACK=1，确认序号 ackseq=x+1。

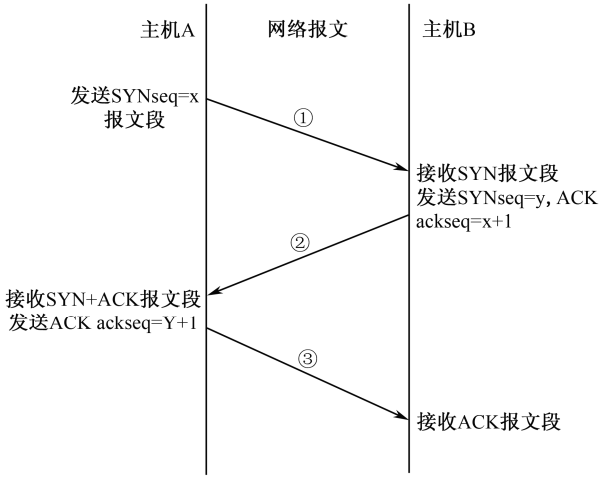


图 3-12　3 次握手的过程

③ 主机 A 确认 B 的同步报文段，建立连接过程结束。

- 发出对端口 2 的确认，ACK=1，确认序号 ackseq=y+1。

这样双方就建立了连接，数据即可双向传输。

3 次握手时，前两个报文段不携带数据，而在第 3 次握手时主机 A 把数据（ackseq=y+1）放在握手的报文段中连同对主机 B 的确认信息（ACK=1, ackseq=y+1）一起发送出去。

为什么要进行 3 次握手呢？是否可以只交换两个报文段呢？实际上，在只交换两个报文的情况下，如果发生异常情况，如有延迟的连接请求报文段突然又传送到主机，就可能产生错误。

考虑这样一种情况：主机 A 发出连接请求报文，但它在某些中间网络节点延迟的时间过长，主机 A 的重传定时时限到，于是又重发一次连接请求，这次 B 收到了连接请求，发回确认，A 收到确认，建立了连接，而且 A 只发送很短的①数据，这样很快就传输完毕并释放了连接。

但是，主机 A 的第一个连接请求报文段没有真正丢失，此时它姗姗来迟，到达主机 B。B 无法鉴别这种情况，误认为 A 又发出一次新的连接请求，于是向 A 发出确认报文段，同意建立连接。A 由于并没有要求建立连接，因此不会理睬 B 的确认，但 B 却以为传输连接就这样建立了，并一直等待 A 发来数据，浪费了主机 B 的资源。可见，两次握手会产生问题。

采用 3 次握手的办法可以防止上述延迟的连接请求引起的不正常现象的发生。由图 3-12 的 3 次握手过程看出，若发生了上述同样的情况，虽然连接已经发生了前两次握手，但主机 A 知道这是不正常的连接，不会继续进行正常的第 3 次握手，而主机 A 发送一个复位报文段清除这一连接，如图 3-13 所示。

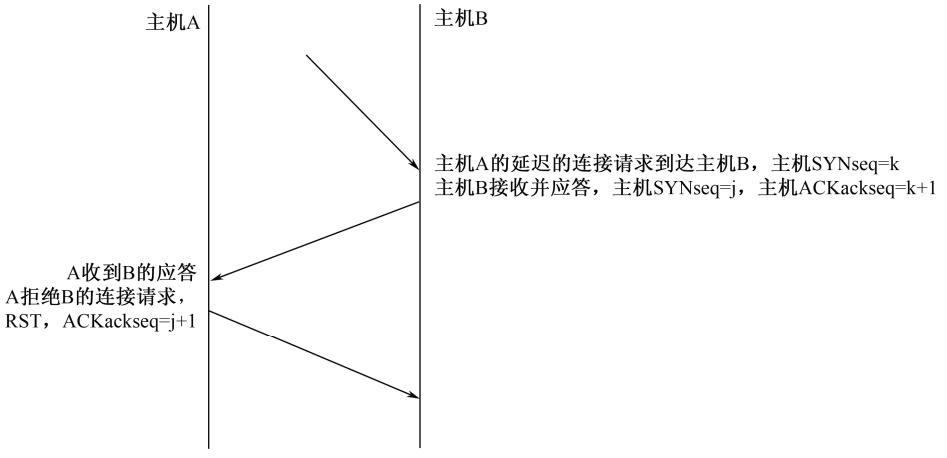


图 3-13　3 次握手清除延迟

在 3 次握手建立连接的过程中，双方 TCP 可以完成以下 3 方面的工作。

1）使每一方都确知对方存在，知道对方已准备就绪。

2）双方确认商定了初始传输序号。每次报文段包括了序号字段和确认序号字段，主机 A 把自己的初始发送序号 x 放到请求连接报文段中送给主机 B。B 收到后记下序号值 x，在对 A 的响应报文中，把自己的初始发送序号 y 写入序号字段，并把 x+1 写入确认序号字段，对 A 进

行确认。在最后一次握手报文中，A 把 y+1 放在确认序号字段，表明期待从 y+1 序号起接收 B 的数据流。

3）双方还可以协商一些通信参数，如通告窗口大小、最大报文段长度 MSS 和窗口比例因子等。

2．关闭 TCP 连接

TCP 的连接是全双工的，可以在两个不同方向上进行数据的独立传输。当某一方（主机 A 或 B）的数据已发送完毕时，TCP 将单向关闭这个连接。此后 TCP 拒绝在该方向上传输数据，但在相反方向上，连接尚未关闭，主机 B 还可以继续发送数据，主机 A 继续接收并进行确认，这种状态称为半关闭（Half-close）状态。

TCP 使用 4 次报文段交互来关闭连接，如图 3-14 所示。

1）主机 A 关闭 A 端口 1 到端口 2 的传输连接。

- 应用程序发完数据，通知 TCP 关闭连接。
- TCP 收到对最后数据的确认后，发送一个 FIN 报文段，FIN=1，seq=x，x 为 A 发送数据的最后字节的序号加 1。虽然是关闭连接，报文段的交换中也要使用序号。

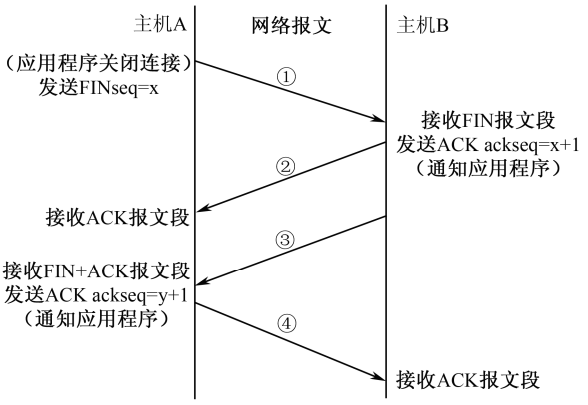


图 3-14　关闭 TCP 连接

2）主机 B 响应连接。

- TCP 软件对主机 A 的 FIN 报文段进行确认，ACK=1，确认序号 ackseq=x+1。
- 通知本端的应用程序：A 方传输已结束。

3）主机 B 关闭端口 2 到主机 A 端口 1 的传输连接。

- 应用程序发完数据，通知 TCP 关闭连接。
- TCP 收到对最后数据的确认后，发送一个 FIN 报文段，FIN=1，seq=y，y 为主机 B 发送数据的最后字节的序号加 1，ACK=1，ackseq=x+1。

4）主机 A 响应。

- TCP 软件对主机 B 的 FIN 报文段进行确认，ACK=1，确认序号 ackseq=y+1。
- 通知本端的应用程序：B 方传输已结束。

此时，双方的全双工连接就彻底关闭了。

首先当发起关闭连接的一方（在图 3-14 中是主机 A）最后发出应答 ACK 后（图 3-14 中的④），将进入一个等待状态 TIME_WAIT，TIME_WAIT 等待定时器设置的时限为两倍的最大报文段生存时间（即 2MSL）。TIME_WAIT 的作用有以下两个方面。

一方面，若主机 A 的应答 ACK 丢失，可以使 A 重发 ACK。如果 A 的 ACK 丢失，主机 B 收不到，当 B 的 FIN 报文段（图 3-14 中的③）的重发定时器超时，B 将重发 FIN 报文段。

当这个报文段到达 A，A 将重发 ACK，并再次启动定时器时限为 2MSL 的 TIME_WAIT 定时器，若这个定时器超时，A 就关闭连接。

另一方面，在等待状态 TIME_WAIT，定义该连接的四元组（本地主机 IP 地址、本地端口号、远程主机 IP 地址、远程端口号）将不能再被使用，只能在 TIME_WAIT 定时器超时后再使用。而定时器超时后，则不会再有该连接上的延迟的连接报文段到来，它们已经因超时被清除网络。这样就防止了未来在建立可能与旧连接有相同序号的新连接时，免遭延迟报文段带来的困扰。之所以使用 2MSL 的定时器时限，是因为第一个 MSL 考虑了一个报文段在一个方向上存活于网络上的最长时间，第二个 MSL 考虑了另一个方向上的应答可以在网络上生存的最长时间。

3．复位 TCP 连接

前面所述的是应用程序传输完数据之后关闭连接，这是正常情况下友好地（Gracefully）关闭连接。但有时也会出现异常情况，不得不中途突然地（Abruptly）关闭连接。TCP 为这种异常的关闭操作提供了复位措施。

欲将连接复位，发起方发出一个报文段，其码元字段的 RST 比特置 1，对方对 RST 报文段的反应是立即退出连接。TCP 要通知应用程序出现了连接复位操作，连接双方立即停止传输并释放这一传输所占用的缓冲区等资源，异常的突然复位可能丢失发送的数据。

3.3.5 TCP 状态机

TCP 的操作可以使用一个具有 11 种状态的有限状态机（Finite State Machine）来表示，图 3-15 描述了 TCP 的有限状态机，图中的圆角矩形表示状态，箭头表示状态之间的转换，各状态的描述如表 3-1 所示。图中用实线表示客户端的状态变迁，用虚线表示服务器的状态变迁。图中的每条状态变换线上均标有“事件/动作”：事件是指用户执行了系统调用（CONNECT、LISITEN、SEND 或 CLOSE）、收到一个报文段（SYN、FIN、ACK 或 RST）或者是出现了超过两倍最大的分组生命期的情况；动作是指发送一个报文段（SYN、FIN 或 ACK）或什么也没有（用“-”表示）。

表 3-1 TCP 状态表

状 态	描 述
CLOSED	关闭状态，没有连接活动或正在进行
LISTEN	监听状态，服务器正在等待执行连接进入
SYN RCVD	收到一个连接请求，尚未确认
SYN SENT	已经发出连接请求，等待确认
ESTABLISHED	连接建立，正常数据传输状态
FIN WAIT 1	（主动关闭）已经发送关闭请求，等待确认
FIN WAIT 2	（主动关闭）收到对方关闭确认，等待对方关闭请求
TIMED WAIT	完成双向关闭，等待所有分组死掉
CLOSING	双方同时尝试关闭，等待对方确认
CLOSE WAIT	（被动关闭）收到对方关闭请求，已经确认
LASTACK	（被动关闭）等待最后一个关闭确认，并等待所有分组死掉

每个连接均开始于 CLOSED 状态。当执行了被动的连接原语（LISTEN）或主动的连接原语（CONNECT）时，它便会脱离 CLOSED 状态。如果此时另一方执行了相对应的连接原语，连接便建立了，并且状态变为 ESTABLISHED。任何一方均可以首先请求释放连接，当连接被释放后，状态又回到了 CLOSED。

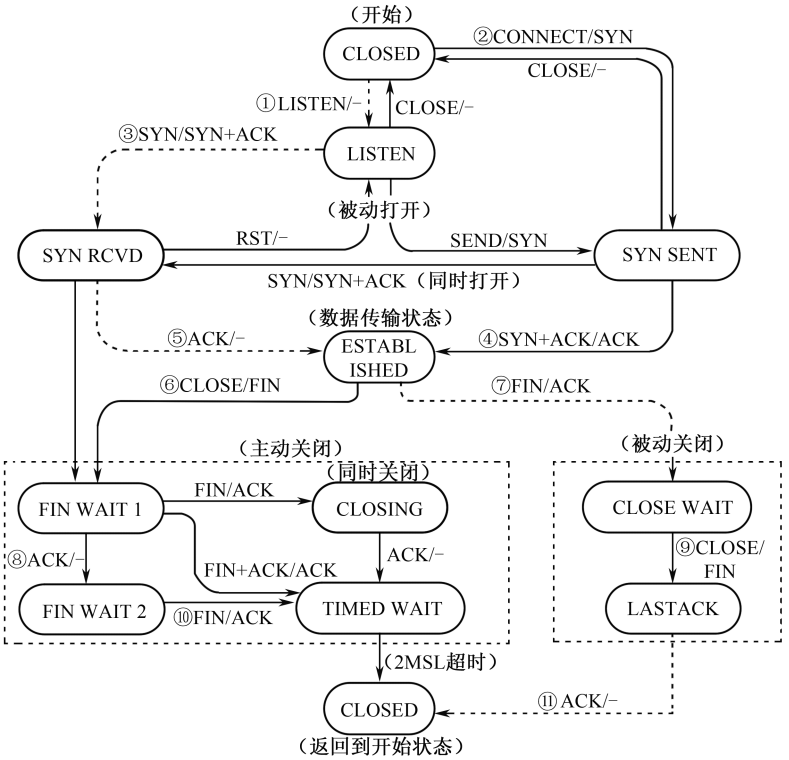


图 3-15　TCP 的有限状态机

1．正常状态转换

图 3-16 所示为显示在 TCP 正常连接的建立与终止中，客户与服务器所经历的不同状态。读者可以对照图 3-15 来阅读，使用图 3-15 的状态图来跟踪图 3-16 的状态变化过程，以便明白每个状态的变化。

1）服务器首先执行 LISTEN 原语进入被动打开状态（LISTEN），等待客户端连接。

2）当客户端的一个应用程序发出 CONNECT 命令后，本地的 TCP 实体为其创建一个连接记录并标记为 SYNSENT 状态，然后向服务器发送一个 SYN 报文段。

3）服务器收到一个 SYN 报文段，其 TCP 实体向客户端发送确认 ACK 报文段，同时发送一个 SYN 信号，进入 SYNRCVD 状态。

4)客户端收到 SYN+ACK 报文段，其 TCP 实体向服务器端发送出 3 次握手的最后一个 ACK 报文段，并转换为 ESTABLISHED 状态。

5）服务器端收到确认的 ACK 报文段，完成了 3 次握手，于是也进入 ESTABLISHED 状态。

在此状态下，双方可以自由传输数据。当一个应用程序完成数据传输任务后，需要关闭 TCP

连接。假设仍由客户端发起主动关闭连接。

1）客户端执行 CLOSE 原语，本地的 TCP 实体发送一个 FIN 报文段并等待响应的确认（进入状态 FIN WAIT 1）。

2）服务器收到一个 FIN 报文段，它确认客户端的请求发回一个 ACK 报文段，进入 CLOSE WAIT 状态。

3）客户端收到确认 ACK 报文段，便转移到 FIN WAIT 2 状态，此时连接在一个方向上被断开。

4）服务器端应用得到通告后，也执行 CLOSED 原语关闭另一个方向的连接，其本地 TCP 实体向客户端发送一个 FIN 报文段，并进入 LAST ACK 状态，等待最后一个 ACK 确认报文段。

5）客户端收到 FIN 报文段并确认，进入 TIMED WAIT 状态，此时双方连接均已经断开，但 TCP 要等待一个 2 MSL，确保该连接的所有分组全部消失，以防止出现确认丢失的情况。当定时器超时后，TCP 删除该连接记录，返回到初始状态（CLOSED）。

6）服务器收到最后一个确认 ACK 报文段，其 TCP 实体便释放该连接，并删除连接记录，返回到初始状态（CLOSED）。

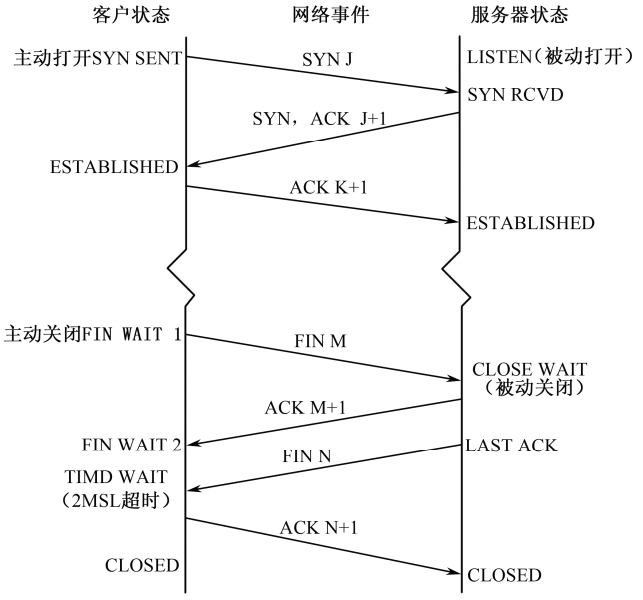


图 3-16　TCP 正常连接建立和终止所对应的状态

2．同时打开

尽管发生的可能性极小，两个应用程序同时彼此执行主动打开的情况还是可能的。每一方必须发送一个 SYN，且这些 SYN 必须传递给对方。这需要每一方使用一个对方知道的端口作为本地端口。例如，主机 A 中的一个应用程序使用本地端口 7777，并与主机 B 的端口 8888 执行主动打开。主机 B 中的应用程序则使用本地端口 8888，并与主机 A 的端口 7777 执行主动打开。TCP 被特意设计，为了可以处理同时打开，对于同时打开它仅建立一条连接，而不是两条连

接（其他的协议簇，最突出的是 OSI 传输层，在这种情况下将建立两条连接，而不是一条连接）。

当出现同时打开的情况时，状态变迁与图 3-15 所示不同。两端几乎同时发送 SYN，并进入 SYN SENT 状态。当每一端收到 SYN 时，状态变为 SYN RCVD，同时它们都再发送 SYN 并对收到的 SYN 进行确认。当双方都收到 SYN 及相应的 ACK 时，状态都变迁为 ESTABLISHED。图 3-17 显示了这些状态变迁过程。

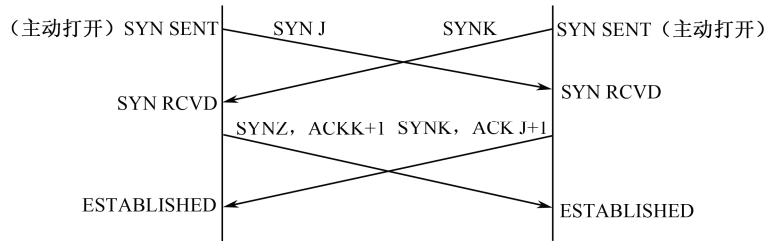


图 3-17　同时打开期间报文段的交换

一个同时打开的连接需要交换 4 次报文段，比正常的 3 次握手多了一个。此外，要注意的是，没有将任何一端称为客户或服务器，因为每一端既是客户又是服务器。

3．同时关闭

正常情况下都是由一方（通常但不总是客户方）发送第一个 FIN 执行主动关闭，但双方都执行主动关闭也是可能的，TCP 也允许这样同时关闭。

在图 3-18 中，当两端应用层同时发出关闭命令时，两端从 ESTABLISHED 变为 FIN_WAIT_1。这将导致双方各发送一个 FIN，两个 FIN 经过网络传输后分别到达另一端。收到 FIN 后，状态由 FIN_WAIT_1 变迁为 CLOSING，并发送最后的 ACK。当收到最后的 ACK 时，状态变化为 TIMED WAIT。图 3-18 总结了这些状态的变化，从图中可以看出同时关闭与正常关闭使用的报文段交换数目相同。

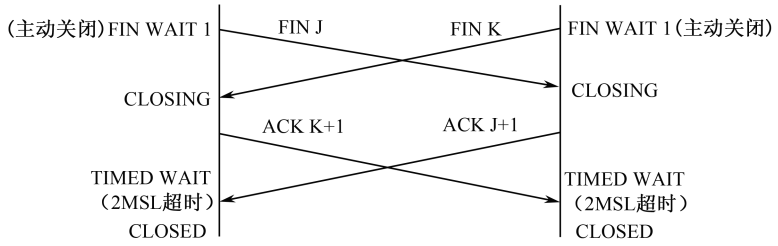


图 3-18　同时关闭期间报文段的交换

4．其他情况

1）服务方打开：从 LISTEN 到 SYN SENT 的变迁是正确的，它由服务器主动发出 SYN 报文段，但 Berkeley 版的 TCP 并不支持它。

2）重置连接（复位）：只有当 SYN RCVD 状态从 LISTEN 状态（正常情况）进入，而不是从 SYN SENT 状态（同时打开）进入时，从 SYN RCVD 回到 LISTEN 的状态变迁才是有效的。

这意味着如果执行被动打开（进入 LISTEN），将收到一个 SYN，发送一个带 ACK 的 SYN（进入 SYN RCVD），然后收到一个 RST，而不是一个 ACK，便回到 LISTEN 状态并等待另一个连接请求的到来。

3）快速关闭：在主动关闭后的 FIN_WAIT_1 状态，如果收到的报文段不仅是 ACK，而且还包括对方的 FIN 信号，则直接进入 TIMED WAIT 状态，向对方发送 ACK 报文段，然后等待超时。

另外，TIMED WAIT 状态的等待超时需要再进行详细解释，因为它直接影响到网络应用程序的表现。

每个具体的 TCP 实现必须选择一个报文段最大生存时间 MSL，它是任何报文段被丢弃前在网络内的最长时间。我们知道这个时间是有限的，因为 TCP 报文段以 IP 数据报在网络内传输，而 IP 数据报又限制其生产事件的 TTL 字段。RFC793[Postel 1981c]指出 MSL 为 2min。然而，实现中的常用值是 30s、1min 或 2min。

对于一个具体实现所给定的 MSL 值，处理的原则是：当 TCP 执行一个主动关闭，并发回最后一个 ACK 时，该连接必须在 TIMED WAIT 状态停留的时间为 2MSL，因此 TIMED WAIT 状态也称为 2MSL 等待状态。在这段时间内，如果最后的 ACK 丢失，对方会超时并重发最后的 FIN，这样本地 TCP 可以再次发送 ACK 报文段（这是它唯一可以发送的报文，并重置 2MSL 定时器）。

这种 2MSL 等待的另一个结果是这个 TCP 连接在 2MSL 等待期间，定义这个连接的套接字（Socket，即客户的 IP 地址和端口号、服务器的 IP 地址和端口号）不能再被使用。这个连接只能在 2MSL 结束后才能再被使用。在连接处于 2MSL 等待时，任何迟到的报文段将被丢弃。

假设图 3-18 中客户执行主动关闭并进入 TIMED WAIT，这是正常的情况，因为服务器通常执行被动关闭，不会进入 TIMED WAIT 状态。这表示如果终止一个客户程序，并立即重新启动这个客户程序，则这个新客户程序将不能重用相同的本地端口。这不会带来问题，因为客户使用本地端口，而不关心这个端口号是什么。然而，对于服务器，情况就有所不同，因为服务器使用周知端口。如果我们终止一个已经建立连接的服务器程序，并试图重新启动这个程序，服务器程序将不能把它的这个周知端口赋予它的端口，因为那个端口是处于 2MSL 连接的一部分。在重新启动服务器程序前，它需要 1～4min。这就是很多网络服务器程序被杀死后不能够马上启动的原因（错误提示为 Address already in use）。确认之后，表示连接建立，可以开始传输数据。

3.3.6 TCP 重传机制

重传机制是为了进行差错控制，是 TCP 可靠性的一个重要措施。每发出一个报文段，TCP 保存该报文段的副本，同时启动一个重传定时器（Retransmission Timer）。重传定时器设置一个超时重传时限（Retransmission Time Out，RTO）。如果该报文段中的数据还没有得到确认，重传定时器的计时值超过 RTO，TCP 就认为该报文段已经丢失或损坏，从而重传该报文段。

局域网数据链路层的控制机制也使用重传定时器。然而，TCP 是针对互联网环境的协议，互联网比单纯的局域网要复杂得多。首先，同一个源站发送的报文段到不同目的站的路径不同，

报文段可能只经过一个时延很小的单跳网络，也可能要经过时延很大的多跳网络，因而所需要的时间大不相同；其次，每个路由器产生的时延与网络负荷密切相关，互联网的负荷经常发生变化，负荷大时可能发送拥塞，网络负荷的变化使报文段在不同时间经过相同的路径所需要的时间也不同。可见，在 Internet 环境中，传输层数据报的往返时间的变化很大。图 3-19 表示了局域网数据链路层往返时间的概率密度和TCP中往返时间的概率密度的分布情况。前者方差很小，重传定时器的 RTO 容易设定，如图中的 T_1。而后者方差很大，如果把 RTO 调小（如图中的 T_2），则会产生大量不必要的重传；如果把 RTO 调大（如图中的 T_3），则一旦分组丢失，过长的重传机制时延会导致网络性能下降。因此，TCP 采用自适应重传算法（Adaptive Retransmission Algorithm）计算 RTO 以适应互联网时延的变化性。

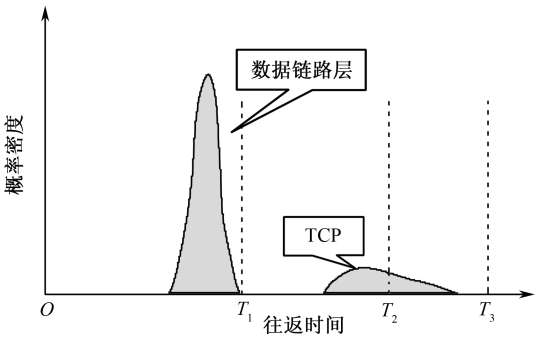


图 3-19　数据链路层与 TCP 往返时间的概率密度

1．往返时间估计与 RTO

TCP 的自适应重传算法随时估算每个连接的传输时延，据此调整重传定时器的定时时限。

对于每次传输，TCP 都记录报文段发送出去的时间和确认返回的时间，由这两个时间值，TCP 计算出报文段往返所经历的时间，称为报文段样本往返时间（Round Trip Time，RTT）。

估计的往返时间称为平滑往返时间（Smoothed Round Trip Time，SRTT），使用式（3-1）进行加权平均可以求出 SRTT，它相当于一个低通滤波器：

$$srtt(k)=\alpha\times srtt(k-1)+(1-\alpha)\times rtt(k) \qquad (3\text{-}1)$$

式中：$0\leqslant\alpha\leqslant1$，建议为 0.8～0.9，实际使用 0.875；$k$ 为计算的步数，$srtt$（k）和 rtt（k）两个变量分别存储第 k 步估计的平滑往返时间和第 k 步测得的样本往返时间。

由式（3-1）不难看出，选用的 α 值越接近于 1，则 SRTT 对短暂的时延变化越不敏感；而 α 值越接近于 0，则 SRTT 对时延变化越敏感，能够越快地随时延变化。

发送分组时，TCP 计算出 RTO，用它设置重传定时器。RTO 是当前的 SRTT 的函数。早期运行的 TCP 使用加权 β（$\beta>1$），使 RTO 大于当前 SRTT：

$$rto(k)=\beta\times srtt(k) \qquad (3\text{-}2)$$

如何确定合适的 β 值呢？在检测到报文段的丢失后，希望能迅速地将该报文段重发出去，这样可以提高网络的吞吐量，减少 TCP 重传之前不必要的等待时间。为此，RTO 要尽可能接近当前的往返时间，即希望 β 接近于 1。但从另一方面考虑，如果 β 接近于 1，当前网络有微小的时延变化引起当前实际往返时间比估计的往返时间略大时，就会导致不必要的重传，这会浪费网络的带宽，因此 β 也不能太接近于 1。最初的规范推荐 β=2。

以上就是最初的自适应重传算法，以后又有了改进的算法，不使用固定的 β。改进的算法引入了实测值 rtt（k）和估计值 $srtt$（k）偏差的平滑值 d（k）：

$$d(k)=\gamma\times d(k-1)+(1-\gamma)\times\left|rtt(k)-srtt(k)\right| \tag{3-3}$$

式中：$0\leqslant\gamma\leqslant1$，实际使用 0.75，$d(k)$ 要设定初值。然后使用偏差的平滑值 $d(k)$ 修正 $srtt(k)$，得到重传时的时限 $rto(k)$：

$$rto(k)=srtt(k)+4d(k) \tag{3-4}$$

其中系数 4 是一个实验得到的数值。

实际使用时，式（3-1）和式（3-3）采用如下等效形式：

$$srtt(k)=srtt(k-1)+0.125(rtt(k)-srtt(k-1)) \tag{3-5}$$

$$d(k)=d(k-1)+0.25(\left|rtt(k)-srtt(k)\right|-d(k-1)) \tag{3-6}$$

式（3-5）和式（3-6）再加上式（3-4）就构成 RTO 的自适应重传实用算法。式中的参数为实际采用的值（$\alpha=0.875,\gamma=0.75$），这样，这 3 个式子的技术都很简单，乘法的运算可以通过简单的移位实现，从而减小了开销。

2．Karn 算法

在出现超时重传时，TCP 计算样本往返时间 RTT 还存在问题。发送方 TCP 生成了一个报文段发送出去，由于重传定时器到时没有收到确认，又重传了一次，之后收到了确认。由于这两个报文段完全相同，确认报文也相同，发送方无法辨认出确认是对原报文段还是对重传报文段。这种现象称为确认的二义性（Acknowledgement Ambiguity）。图 3-20（a）表示主机 A 的数据在传输中丢失，重传定时器也超时后重传；图 3-20（b）表示主机 B 对原报文段的确认 ACK 延迟，主机 A 的重传定时器也超时后重传。主机 A 都无法分辨出收到的确认是对原报文段还是对重传报文段。如果认为是对原报文段的确认，图 3-20（a）的情况就不对；如果认为是对重传报文段的确认，图 3-20（b）则不对。

如果认为确认是对原来的报文段，对于图 3-20（a）的情况，计算出的 RTT 值比实际值大，那么当互联网频繁丢失报文段时，会使 SRTT 不断地增长。这使得后面的传输 TCP 设定的 RTO 增大，降低了网络的传输效率。如果认为确认是对最近的报文段，对于图 3-20（b）的情况，则会使计算出的 RTT 比实际值小，使得 SRTT 减小。这使得在后面的传输中，TCP 设定的 RTO 减小。RTO 的减小又进一步加剧了重传的发生，浪费了网络带宽。

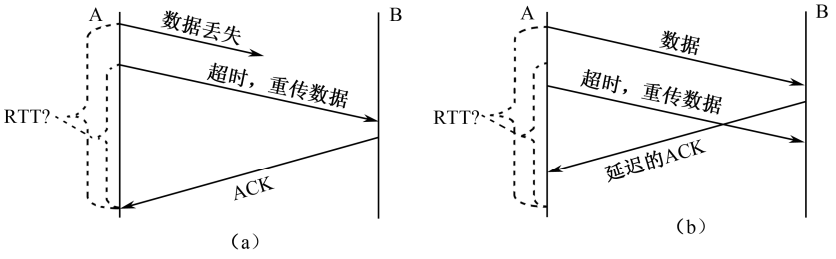


图 3-20　确认的二义性

可见，不论认为 ACK 是对原报文段还是对重传报文段，计算 RTT 都会存在问题。

业余无线电爱好者 Phil Karn 提出了一个实用的解决方法，避免确认的二义性所带来的问题。在式（3-1）中，TCP 不使用重传报文段的样本，而只使用发送一次的报文段的样本，用这

些往返时间 RTT 对估计值 SRTT 进行调整。因此，它避免了确认二义性的问题。

这种简单的 Karn 算法也带来了新的问题，因为它忽略了重传对往返时间的影响。出现重传，意味着网络传输延时加大，应该加大 SRTT 和 RTO。如果继续使用原来没有重传时计算的、小于目前实际情况的 RTO 值，将会使重传继续下去。因此，应该利用重传的信息对 RTO 进行调节。

针对这种情况，Karn 算法使用定时器补偿（Timer Backoff）策略把超时重传的影响估计在内。定时器补偿策略仍然使用上面的公式来计算 RTO 的值，但是每当出现超时重传时，TCP 就使用式（3-7）加大 RTO：

$$rto(k) = \delta \times rto(k-1) \tag{3-7}$$

式中：δ 是一个常数因子，它的典型值是 2。为了避免定时时限的无限增加，在 TCP 的实现中可以规定 RTO 的上限。

总之，为了避免重传时二义性，并使重传定时器的 RTO 值适应网络时延的变化，Karn 算法的思路是：计算往返时间估计值时，忽略重传报文段的样本，当出现超时重传时，要使用定时器补偿策略。实践表明，Karn 算法在分组丢失率很高的网络上也能很好地工作。

最后，关于 RTT 的计算还要做以下两点说明。

（1）由于 TCP 采用的是对数据流的计算的累计确认，发送数据报和确认报文之间通常并没有一一对应的关系。例如，TCP 收到了两个有序的报文段而只发回了一个确认。此时，RTT 的计算采用确认到达的时间与第一个报文段的发送时间的差，因此发送 TCP 在进行 RTO 计算时，应该将延迟确认时间也考虑在内。

（2）后来定义的时间戳选项可以用于 RTT 的计算。使用时间戳选项，确认报文中可以回送报文段的时间戳，标明是对哪个报文段的确认，建立发送报文段和确认报文之间的联系，使发送方便于计算 RTT。这样，可以允许发送方 TCP 为每个确认计算 RTT。同样，由于累计确认方式使发送报文段和确认报文之间没有一一对应的关系。为了正确地计算 RTT，不同情况下确认报文的回送时间戳应该合理地选用某个已到达报文段的时间戳。

3.3.7　TCP 流量控制

我们在第 2 章介绍了数据链路层的流量控制机制。流量控制机制用来保证发送方的数据在任何情况下都不会“淹没”接收方的接收缓冲区，而且还应该使传输达到理想的吞吐量。接收方知道自己接收缓冲区的状况，由接收方控制发送方的数据流量，乃是计算机网络中流量控制的一个基本思路。这一思想不仅适用于数据链路层，而且也适用于传输层。Internet 传输层的 TCP 实现的是端到端（End To End）的流量控制。

1．可变窗口与流量控制

TCP 使用可变滑动窗口机制进行端到端流量控制。

图 3-21 所示为 TCP 发送方滑动窗口示意图。窗口中有 3 个指针：位于滑动窗口左边界的第一个指针把已经发送并得到确认的字节与尚未得到确认的字节区分开来；第二个指针标出了窗口的右边界，指出序列中可以发送的最高字节的序号；第三个字节指针位于窗口的内部，它

划分出已经发送的字节和尚未发送的字节之间的界限。TCP 软件可以不加延迟地发送窗口内的字节，窗口内的指针会随之从左向右移动。

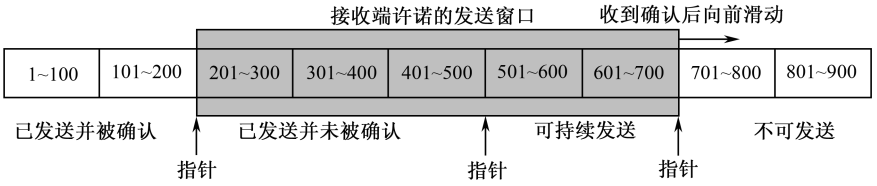


图 3-21　TCP 发送方滑动窗口示意图

由图 3-21 可知，发送方要发送的数据共有 9 个报文段，每个报文段长 100B，而接收方通告的窗口大小为 500B。发送窗口当前的位置表示序号为 1～200 的两个报文段已经发送过，并已收到了收方的确认。在当前收方许诺的窗口大小下，发送方可以在未收到确认前连续发送序号为 201～700 的 500B。假定发送方已发送了 201～500 的 300B，但未收到确认，那么它还可以发送 501～700 的 200B。

如果发送方收到收方发来的确认，就可将发送窗口向前移动。例如，此时发送方收到了收方已正确收到 201～400 的确认，那么发送窗口就向前滑动 300B，701～900 又落入了发送窗口之中，窗口内的数据为 401～900，其中 401～500 是已发送但未收到确认的，发送方此时可发送 501～900 的数据。在收方也有一个和发送方类似的接收窗口，把接收到的字节序列拼接到一起。

在数据链路层的滑动窗口机制中，数据是按帧编号的，而在 TCP 滑动窗口机制中，数据是按字节编号的，如图 3-21 所示。

TCP 使用可变滑动窗口机制有效地进行流量控制。接收方使用确认报文中的通告窗口（Advertisement Window）字段，通告发送方调整发送窗口的大小。此外，写入通告窗口中的数值反映了接收方当前可用接收缓冲区的大小，即它还具有多大的接收能力，并通过确认报文将这一信息反馈给发送方。发送方发送窗口的大小在向前滑动时根据收方的反馈信息进行调节，因而控制了发送数据的流量。

当收方的缓冲区有很大的余量容纳更多数据时，它就发出一个大的通告窗口值，使发送方发送窗口变大，增加发送数据；反之，就发出一个小的通告值，减小发送流量。

在极端情况下，接收缓冲区已经饱和，收方 TCP 使用 0 通告值停止连接上的数据流。当接收缓冲区又有空间可用之后，收方用一个非 0 的通告值激活数据流。TCP 使用 0 通告值停止数据流的一个例外，是允许发送 URG 比特置 1 的用于紧急数据通知的报文段。

图 3-22 是一个简单的例子，说明通过可变的滑动窗口进行流量控制的过程，此例中假定 TCP 可接失序的报文段。主机 A 向主机 B 发送数据，假设每一个报文段长度是 100，ISN 为 1，双方开始商定的窗口大小是 400B，图中的序号是每个报文段的。数据发送和确认的流程共包括 13 步，其中 4 次改变窗口大小以调节发送流量。其中图 3-22（a）是发送站 A 的发送窗口变化情况，窗口中的指针指示第几步之后已发送出去但尚未收到确认的报文段与未发送报文段之间的分界。图 3-22（b）是发送报文段和确认报文段往返的传输过程。

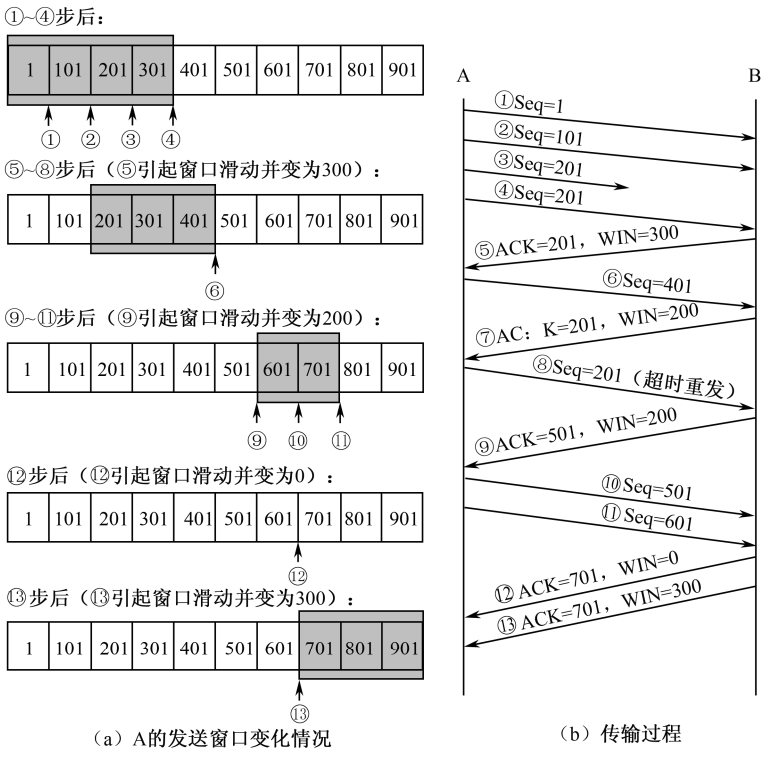


图 3-22　TCP 可变窗口流量控制

上文曾讲到，当收方接收缓冲区满时，可以使用 0 通告窗口值停止 TCP 连接上的通信流量。但实际使用中，滑动窗口的 0 通告值可能带来一个问题。考虑下述情况：接收方发出了一个 0 通告窗口，发送方将发送窗口调整为 0，暂停发送并等待。之后应用程序从缓冲区取走了数据，使缓冲区有了空间，接收方发送一个非 0 的窗口通告，通知发送方又可以发送数据了。但不幸的是，这一非 0 通告的报文丢失了，发送方和接收方都等待对方的动作，因而造成了死锁。

解决这种现象的办法是 TCP 使用坚持定时器（Persistence Timer）。当接收到 0 通告值的确认后，发送方启动坚持定时器，当定时器设定的时间到，发送方发送一个探测报文段。收方对探测报文段的响应包含了通告窗口的通告值。如果通告值不为 0，则发送方调整发送窗口进行发送；若通告值为 0，则重新设定坚持定时器重复上述过程。

2．糊涂窗口综合征及其对策

TCP 用可变窗口进行流量控制是一个行之有效的措施，但也还有一些问题需要进一步研究。例如，在应用程序从接收缓冲区取走多少数据后接收端 TCP 更新通告窗口？是不是应用程序送数据到发送窗口后发送端 TCP 就马上发出去？TCP 如何掌握发送时机？这些问题 TCP 标准并没有明确规定，而是留给实现软件去解决。

早期 TCP 研究人员发现，在发送端或接收端的应用程序工作速度很慢时，TCP 会出现短报文段传输的问题，影响了网络的传输效率。

首先看接收端的应用程序工作速度很慢的情况。假设接收方应用程序每次仅能读取 1B。在

建立了传输连接之后，发送方应用程序快速生成了数据，发送方 TCP 软件传输的报文段很快会装满收方的缓冲区，发送方 TCP 得到确认，得知整个接收窗口已饱和，通告窗口为 0，暂时无法发送后续数据。当收方应用程序从已饱和的缓冲区读取了 1B 后，缓冲区就有了 1B 的可用空间，TCP 软件生成一个确认，其通告窗口为 1B。发送方 TCP 得到确认后，会发送包含 1B 数据的报文段。收方应用程序读取了下一字节后，TCP 又发回了通告窗口为 1B 的确认，这又使发送方 TCP 又发送了含有 1B 数据的报文段。最终形成了一个稳定的传输，但每次传输的报文段仅包含 1B 的数据，而总的 IP 数据报却有 41B，首部和净荷之比如此悬殊，大大降低了网络的传输效率。上述问题，即每个确认报文通告了小的窗口，使每个报文段仅携带少量的数据，这种现象称为糊涂窗口综合征（Silly Window Syndrome，SWS）。以上 SWS 现象如图 3-23 所示。

再看发送端应用程序工作很慢的情况。例如，一个 Telnet 连接，受限于人工键盘操作的速度，发送方应用程序每次可能只生成 1B 的数据，发送方 TCP 软件每次发送仅包含 1B 数据的短报文段。

可见，当发送应用程序产生数据很慢，或接收应用程序接收数据很慢，甚至两者兼有，TCP 将发生短报文段传输问题，即 SWS，大大降低了网络的传输效率，应该予以解决。

收方应对 SWS 的策略有两种方法。

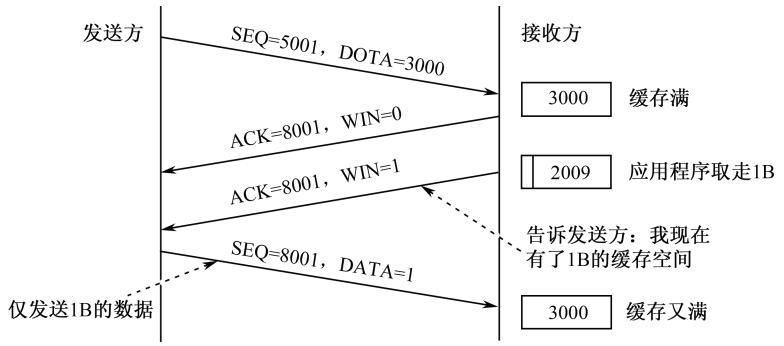


图 3-23　接收应用程序工作慢引起的 SWS

1）Clark 方法。接收缓冲区满之后，发出 0 通告窗口，此后应用程序取走下一（或少量）字节时，TCP 并不发回通告窗口为 1（或少量）字节的确认，而是等待应用程序逐步取走数据，使缓冲区可用空间达到 MSS 或缓冲区总空间的一半之后，才更新通告窗口。

2）延时确认。TCP 延迟一段时间后再发送确认。

发送方应对 SWS 的策略称为 Nagle 算法。Nagle 算法工作过程如下。

（1）当应用程序产生第一个数据块时，即使只有 1B，TCP 也会立即发送出去。

（2）在发出第一个报文段后，发送端 TCP 在输出缓冲区积累数据，并等待下列事件之一发生，触发一次新的发送。

- 收到接收端 TCP 发送的一个确认。
- 数据已积累一个 MSS，这使得当应用程序生成数据的速率比较快时，发送的报文段将包含足够多的数据，从而达到较大的传输流量。

（3）以后的传输，重复步骤（2）。

3.3.8　TCP 拥塞控制简介

拥塞是由 Internet 中的分组转发节点超载引起的，是分组交换网络共同的问题，在 Internet 中使用无连接的分组交换技术，拥塞控制十分重要。

产生拥塞的原因是网络中一个或若干个路由器的数据报负载相对它的处理能力过重，过多的数据报只能在路由器的缓存队列中排队等待转发，造成严重的传输时延。路由器的缓存能力总是有限的，严重情况下，数据报将充满缓存，于是路由器不得不丢弃数据报。

增大路由器的处理能力和缓存空间对解决拥塞是有益的。处理能力越大越好，但缓存空间并非如此。过多的缓存空间虽然可能容纳更多的数据报而不至于被丢弃，但增加了传输时延。源站一般不知道因何原因或在何处发生拥塞等细节，对它来说，拥塞表现为时延增加。TCP 使用超时重传机制，传输时延增加会引起数据报的重传，重传增加了通信流量，通信流量增加又进一步加剧了重传，如此恶性循环，可能导致拥塞崩溃（Congestion Collapse），网络吞吐量变得很小。

和流量控制一样，TCP/IP 拥塞控制最根本的措施是减慢源发站的发送速率，即源抑制（Source Quench）技术，这由传输层的 TCP 来实现。

但拥塞涉及整个通信子网，网络层也有一些措施抑制拥塞，路由器直接面对拥塞，它也参与拥塞控制。下面先简单介绍路由器的拥塞控制。

1．路由器的拥塞控制

由第 2 章可知，路由器要周期性地测试输出队列，检测拥塞的发生。发生拥塞后，一方面向发送站发出 ICMP 源抑制报文，报告拥塞消息；另一方面，采取分组丢弃策略抑制拥塞。

路由器的分组丢弃策略是指在输出队列缓冲区没有空间时，丢弃部分分组，好的丢弃策略有助于避免拥塞。对于文件传输，旧分组比新分组更有价值，因为丢弃旧的分组可能导致更多的分组超时重传，因此可以丢弃新的分组，这种策略认为旧的比新的好，称为葡萄酒策略。相反的策略认为新的比旧的更有价值，如多媒体信息，称为牛奶策略。

路由器分组丢弃的一种改进策略称为随机早期检测（Random Early Detection，RED）算法，路由器实现主动的队列管理（Active Queue Management），使路由器保持较小的平均队列长度，有序地吸收突发流，不必等到溢出时丢弃较多的分组。另外，较小的平均队列长度也可以减小分组端到端的传输时延。

RED 认为，Internet 的流量有很大的突发性，瞬时队列长度在短时间内可能变化很大，这是不可避免的，用它推断拥塞不是很恰当。RED 使用分组平均队列长度 L_a 而不是当前的瞬时分组队列长度 L_c 去决策分组丢弃。当队列不空时，$L_a=(1-\omega)L_a+\omega L_c$，其中 ω 为权系数。RED 对分组平均队列长度设置了上、下限，当 L_a 小于下限时，不丢弃分组；当 L_a 大于上限时，丢弃到达的分组；当 L_a 介于上、下限之间时，按概率 P 丢弃分组，概率 P 是 L_a 和 n 的函数，其中 n 是上次丢弃分组之后到达的分组数，L_a 和 n 的值越大，概率 P 越大。

2．TCP 拥塞控制

早期的 TCP 使用累计确认、超时重传和回退-N ARQ 等机制，对网络拥塞几乎没有进行控制。1986 年，Internet 出现了吞吐量突然迅速下降的拥塞崩溃现象，开始在 TCP 中增加拥塞控制算法。

为了避免和控制拥塞，TCP 推荐使用以下几种技术：慢启动（Slow Start）和拥塞避免（Congestion Avoidance），后来又提出了快速重传（Fast Retransmission）和快速恢复（Fast Recovery）。

使用这些技术的前提是认为绝大多数的报文丢弃都由拥塞导致，这是基于以下事实：目前的通信技术已使得由于通信线路问题引起的误码而造成报文丢失的概率很小。

TCP 的拥塞控制机制是闭环控制，它要通过直接或间接地反馈信息了解当前网络的拥塞状况。TCP 可以通过一些途径发现拥塞发生：报文段的超时重传及来自 ICMP 的源抑制报文的信息。

如前所述，通告窗口是接收方根据接收的情况许诺的发送窗口值，是接收方反馈的流量控制信息。为了进行拥塞控制，TCP 又设置了另一个窗口，即拥塞窗口（Congestion Window），它用于发送方的流量控制。当发生拥塞时，拥塞窗口将数据流量限制为小于接收方的接收能力。在未发生拥塞的稳定运行情况下，拥塞窗口和通告窗口是一致的。

发送方发送数据时，既要考虑到接收方的接收能力，又要考虑到拥塞状况，因此，发送窗口按式（3-8）选择通告的拥塞窗口和接收窗口中较小的一个。

$$swnd = \min(cwnd, rwnd) \tag{3-8}$$

式中：变量 *swnd*、*cwnd* 和 *rwnd* 分别为发送方的发送窗口、接收方通告的拥塞窗口和接收窗口。

为了进行拥塞控制，对每个 TCP 连接设置了 *cwnd* 变量，还设置了另一个变量：慢启动门限（Slow Start Threshold），它用来分界慢启动和拥塞避免策略，用变量 *ssthreshold* 表示。TCP 规定：当 *cwnd*<*ssthreshold* 时，使用慢启动策略；当 *cwnd*>*ssthreshold* 时，使用拥塞避免策略；当 *cwnd*=*ssthreshold* 时，可以使用慢启动策略，也可使用拥塞避免策略。

慢启动和拥塞避免策略是早就提出来的拥塞控制策略，我们结合图 3-24 的示例进行具体说明。图 3-24 所示的拥塞控制过程中，曲线的横坐标是传输次数。发送方发完本次发送窗口中的全部报文段并都收到了确认，称为“传输 1 次”。在 *swnd*=*cwnd*<*ssthreshold* 的情况下，传输 1 次是发送方连续发完发送窗口中的全部报文段并都收到了确认，所用的时间大约是 RTT，报文段发送用时比在传输线路上传输用时一般要少得多。

拥塞控制的过程如下。

1）当对一个 TCP 连接初始化时：

① 设置拥塞窗口初值，不能大于两个报文段长度，一般是 *cwnd*=1MSS（B）。为方便计算，下面使用报文段为单位。在图 3-24 中，设置拥塞窗口初值 *cwnd*=1。一个 TCP 连接上的传输也从慢启动开始，可以探测可用的网络带宽。

② 设置慢启动门限初值，没有具体地规定初值的大小，很多 TCP 实现使用收方通告的值。在图中，设置慢启动门限初值 *ssthreshold*=16。

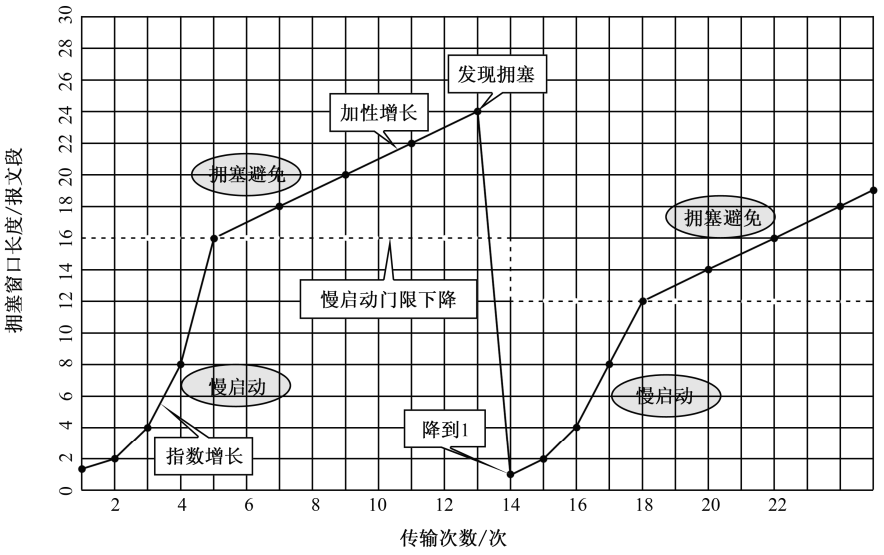


图 3-24　拥塞控制示例

2）TCP 开始发送过程，发送窗口 *swnd* 按式（3-8）计算，一般地，通告的接收窗口 *rwnd* 足够大，它限制了 *swnd* 的上限不超过 *rwnd*，*swnd* 实际上等于 *cwnd*。

3）每次传输都调节一次拥塞窗口，进而调节了发送窗口，调节方式如下。

① 当 *cwnd*<*ssthreshold* 时，执行慢启动。*cwnd* 从初值 1 开始，每收到一对新报文段的确认 ACK，*cwnd*=*cwnd*+1。这样，第 1 次传输完，收到 1 个 ACK，*cwnd* 增加到 2；第 2 次传输完，收到两个 ACK，*cwnd* 增加到 4……因此 *cwnd* 按指数规律增长：1→2→4→8→16，即每传输 1 次，*cwnd* 加倍，图 3-24 中，TCP 从开始到第 4 次传输，执行慢启动过程。

② 当 *cwnd*>*ssthreshold* 时，转入拥塞避免。*cwnd* 从 *ssthreshold* 开始，每收到一个对新报文段的确认 ACK，*cwnd*=*cwnd*+1/*cwnd*。这样，第 1 次传输完，收到 1 个 ACK，*cwnd* 增加到 1。此时，发送窗口减慢增加速度，变为加性增长（Additive Increase）。图 3-24 中，当第 4 次传输完时，*cwnd* 达到设定的慢启动门限值 16，之后，*cwnd* 变为加性增长，直到再次检测到拥塞。在 *cwnd* 加性增长的过程中，*swnd* 要受到式（3-8）的约束，最大到 *rwnd*。

4）如果在某时刻发生了拥塞，则置 *ssthreshold*=max（*swnd*/2，2），即将 *ssthreshold* 降到拥塞发生时 *swnd* 的一半，但不能小于 2，并令 *cwnd*=1，即拥塞发生后，慢启动的 *cwnd* 初值为 1，开始慢启动过程。图 3-24 中，当进行到第 13 次传输时，重发定时器出现超时，发生了拥塞。此时 *cwnd* 已增长到 24（假设 *rwnd*＞24），那么此时 *swnd* 也是 24，于是 TCP 令 *ssthreshold*=*swnd*/2=12，*cwnd*=1，*cwnd* 回到慢启动的起点，TCP 又进入慢启动过程。

从以上过程可以看出，“慢启动”指每出现一次拥塞，拥塞窗口都要降到 1 个报文段长度的起点，然后增加，使报文段从最小的流量开始注入到网络之中。不过慢启动这个名词并不十分确切，因为拥塞窗口按指数增长，增长的速率并不慢。实际上它是“低启动”，拥塞发生之后，拥塞窗口都要设置为 1 个报文段长度，传输过程从这样的低起点开始启动。

“拥塞避免”是指当拥塞窗口增大到慢启动门限值之后，连接上的流量已增大到一定的程度，

就将拥塞窗口增长速率由指数增长变为加性增长，以避免再次出现拥塞。

后来人们又提出了拥塞控制算法的修改建议，即快速重传和快速恢复技术，它们一般一起被使用。

在 TCP 发生报文丢失后，接收方会收到失序的报文段。收方的响应是立即发出一个确认，让发送方知道收到一个失序的报文段，并告诉对方自己希望收到的序号，即丢失报文段的序号。在该报文段没有收到的情况下，接收方每收到一个后续的报文段，都会尽快发出一个重复的确认。因此，根据接收到的重复确认报文信息，也可以发现丢失或滞留在网络中的报文段。

快速重传和快速恢复算法如下。

1）3 个重复的确认（共 4 个 ACK）后，令 *ssthreshold*=max（*swnd*/2，2），和慢启动的算法一样。

2）重传报文段，并令 *cwnd*=*ssthreshold*+3。这和慢启动算法置 *cwnd* 为最小值 1 不一样。收到 3 个重复的 ACK，说明有 3 个报文段已经离开了网络进入收方缓冲区，将 *cwnd* 在 *ssthreshold* 基础上又加 3。

3）收到一个重复的确认时，令 *cwnd*=*cwnd*+1。重复的确认说明又有一个报文段已离开网络，将 *cwnd* 加 1。

4）若发送窗口允许，就发送一个新的报文段。

5）当新的非重复的确认（这是对第（2）步重传报文段的确认，重传报文段已到收方）到达时，令 *cwnd*=*ssthreshold*（第（1）步 *cwnd*=*swnd*），然后使用拥塞避免算法。

以上算法中，当收到第 3 个重复的确认时，就认为报文丢失而重传报文段，而不必等到重传定时器到时，故称快速重传。接着取消启动而执行快速恢复，并不把 *cwnd* 降到 1。取消执行慢启动的原因是由于收方的重复确认，不仅仅告诉发送方一个报文段已丢失，而且还表明收方正在成功地接收失序的报文段，报文段已经离开网络进入了收方的缓存，TCP 连接上仍然有数据流在传输，因而不必执行慢启动锐减数据流。

快速重传算法最早出现在 4.3BSD Tahoe TCP 版本，它包括慢启动、拥塞避免和快速重传算法。1990 年后，4.3BSD Reno TCP 版本中，使用快速重传和快速恢复算法，在建立连接和超时重传时使用慢启动和拥塞避免算法。Reno TCP 是大多数 TCP 实现采用的拥塞控制算法。

当一个发送窗口中有多个报文段丢失时，上述快速重传和快速恢复算法不能快速恢复。针对这种情况，对算法又进行了改进[RFC2582]。实现改进算法的 TCP 版本称为 New-Reno TCP，它的 RFC 还是实验性的。

3.3.9 TCP 差错控制

TCP 是一个可靠的传输层协议，发送端的应用进程将数据流交付给 TCP，它将依靠 TCP 将整个数据流交付给接收端的应用进程，并且按顺序、没有差错，也没有丢弃或重复。差错控制包括检测损伤程度的报文段、丢失的报文段、失序的报文段和重复的报文段，以及检测出差错后纠正差错的机制。由于 TCP 建立在无连接、不可靠的 IP 基础上，因此 TCP 只能通过差错控制来提供可靠性。TCP 的可靠性问题包括数据丢失后恢复及连接的可靠建立问题。

TCP 的差错检测通过 3 种简单工具完成：校验和、确认和超时。每个 TCP 报文段都包括校验和字段。校验和用来检查报文段是否出现传输错误。如果报文段出现传输错误，TCP 检查出错就丢弃该报文段。发送端 TCP 通过检查接收端的确认，判断发送的数据报是否已正确到达目的地。如果发出的报文段在超时规定的时间没有收到确认，发送端将判断该报文段丢失或传输错误。

1．传输出错报文段的处理

传输出错报文段又称受损伤的报文段。图 3-25 给出了受损伤的报文段到达目的进程的过程。在这个例子中，源进程发送报文段 1～3，每个报文段的数据长度为 200B，序号从报文段 1201～1800。接收 TCP 收到报文段 1 和 2 后，向发送端发送确认号 1601 的确认报文段，表示它已正确接收，序号为 1201～1600，并且希望下次接收序号为 1601。如果接收端发现报文段 3 受到损伤，因此丢弃报文段 3，目的进程没有对该报文段进行确认。若为报文段 3 设置的接收确认时间到，发送端 TCP 认为接收确认超时，重传报文段 3。在正确接收到报文段 3 后，目的进程发送对序号为 1801 的确认信息，表示它正确和完整地收到序号为 1201～1800 的报文。

2．丢失的报文段

图 3-26 给出了丢失的报文段到达目的进程的过程。从源进程和目的进程的角度来看，丢失的报文段与受损伤的报文段是一样的。受损伤的报文段是被目的进程丢弃的，丢弃的报文段是被某一个中间节点丢失的，并且不会到达目的进程。

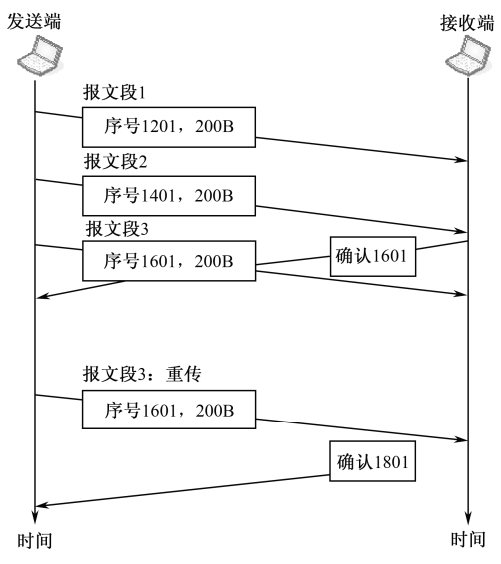


图 3-25　受损伤的报文段到达目的进程的过程

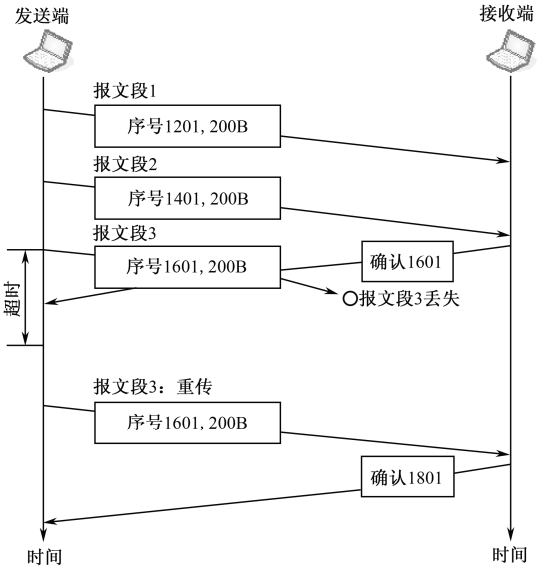


图 3-26　丢失的报文段到达目的进程的过程

3．重复的报文段

重复的报文段一般是由发送端 TCP 造成的。当超时截止期到而没有收到确认时，发送端的

TCP 就会重复发送该报文段。接收端的 TCP 处理重复报文段很简单。接收端的 TCP 判断有同样序号的报文段到达时，丢弃重复的报文段。

4．乱序的报文段

TCP 使用无连接的 IP 服务。TCP 报文段封装在 IP 数据报中。每一个 IP 数据报都独立地由路由器来找到合适的传输路径。由于不同的数据报可以沿着不同的路径到达，那么同一个数据报的多个 TCP 报文段也就有可能不按顺序到达目的主机，出现乱序现象。接收端的 TCP 处理乱序报文段的方法很简单，对乱序的报文段不确认，直到收到所有它以前的报文段为止。当然，如果确认超时，发送端的 TCP 发送报文段的确认计时器会因超时而重新发送该报文段。

5．确认丢失

图 3-27 给出了确认丢失的过程与处理方法。确认是由接收端进程发出的，可能在传输过程出现丢失。TCP 的确认机制采用了累计确认办法。例如，发送端连续发送序号为 1201、1401、1601 的报文段 1、2、3。接收端在正确接收第 1、2 个报文段后，发送确认号为 1601 的确认报文段；在接收第 3 个报文段时，发送确认号为 1801 的确认报文段。但是确认号为 1601 的确认报文段丢失，而在报文段 1 确认超时之前，收到确认号为 1801 的确认报文，则根据累计确认方法，只要收到 1801 的确认报文，就说明序号为 1801 之前的所有字节均被正确接收，发送端可以忽略之前的确认丢失。

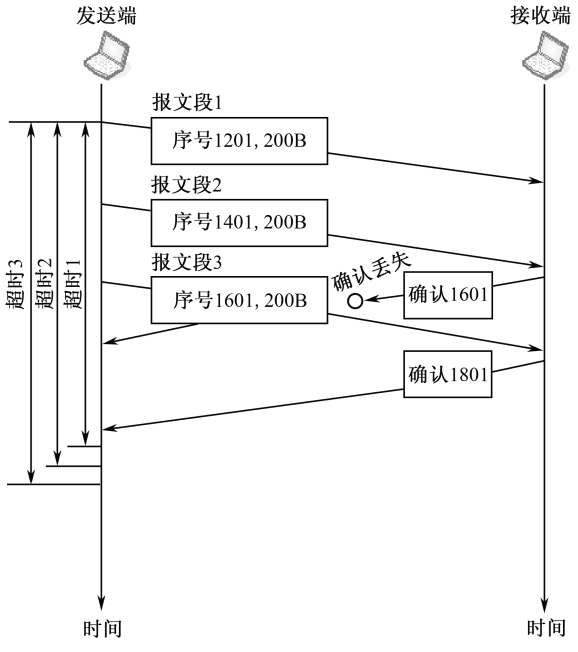


图 3-27　确认丢失的过程与处理方法

由于 TCP 连接能够提供全双工通信，因此在实际的通信过程中，通信双方都不需要专门发送确认报文段，可以在传送数据的同时以“捎带确认”方法完成确认过程，这样做可以提高系统工作效率。

3.3.10　TCP 的计时器

为了实现 TCP 的功能，TCP 使用了 4 种计时器：重传计时器、坚持计时器、保持计时器和时间等待计时器。

1．重传计时器

为了控制丢失的或丢弃的报文段，TCP 采用了重传计时器，用来计算确认到达时间，决定报文段是否需要重传及重传的时间。当 TCP 发送报文段时，它就创建该报文段的重传计时器。此后可能会有两种情况发生：如果计时器截止时间到之前收到对该报文段的确认，则撤销此计时器；否则，TCP 重传该报文段并计时器复位。

由于建立 TCP 连接的两个主机可能在同一网络中，也可能相隔了数千个互连的网络，因此传输经历的路径长度可能相差很大，这表明 TCP 不能对所有连接使用相同的重传时间。一个网络中的重传时间也可以不固定，这要取决于该网络通信量的大小。

2．坚持计时器

为了处理零窗口大小通知，TCP 需要使用坚持计时器。假设接收端的 TCP 宣布窗口大小为零，发送端的 TCP 就停止传送报文段，直到接收端的 TCP 发送确认并宣布一个非零的窗口。这个确认可能会丢失，如果确认报文丢失，接收端的 TCP 就认为它已经完成任务，并等待着发送端的 TCP 发送更多的报文段。发送端的 TCP 由于没有收到确认，就等待对方发送确认来通知窗口大小。这样，双方的 TCP 都永远等待着对方的通知状态，这就可能出现死锁。

为了避免出现死锁，TCP 为每一个连接使用一个坚持计时器。当发送端的 TCP 收到一个窗口大小为零的确认时，就需要启动坚持计时器。当坚持计时器期限到时，发送端的 TCP 就发送一个特殊的探测报文段，这个报文段只有 1B 的数据。探测报文段提醒接收端的 TCP，确认已经丢失，必须重传。

坚持计时器的值一般与重传计时器的数值相同。如果发送端没有收到从接收端来的响应，则需要将坚持计时器的值加倍和复位，并继续发送探测报文段。坚持计时器的值最大到门限值（通常是 60s）。此后，发送端每隔 60s 就发送一个探测报文段，直到窗口重新打开。

3．保持计时器

保持计时器又称激活计时器，它用来防止在两个 TCP 之间的连接处于长期空闲状态。如果客户建立到服务器的连接，发送一些数据，然后停止传送，可能这个客户端出现故障。在这种情况下，这个连接将永远处于打开状态。为了防止出现这类问题，服务器需要设置保持计时器，服务器收到客户的信息后就将计时器复位。如果服务器超过保持计时器规定的时间还没有收到客户的信息，就每隔一个时间（如 75s）发送一个探测报文段。如果发送 10 个探测报文段后没有响应，就判断客户端出现故障，服务器将终止该连接。

4．时间等待计时器

时间等待计时器在释放传输连接期间使用。当 TCP 释放一个连接时，它并不认为这个连接马上就真正地关闭。在时间等待期间中，连接还处于一种过渡状态。当被动释放传输连接的一方发出同意释放传输连接的报文段时，打开时间等待计时器。如果时间等待计时器超时之前接到应答报文段，则正确释放传输连接。如果出现超时，则重传同意释放传输连接的报文段。

3.3.11 TCP 的安全

1．TCP 序列号猜测攻击

TCP 序列号猜测攻击是一种非常有名的欺骗攻击方法，攻击者试图通过猜测出要攻击的系统用于下一次连接时所用的 ISN（初始序列号）来达到欺骗攻击对象的目的。实际上，对 TCP 序列号的猜测攻击不仅可以在建 TCP 连接的 3 次握手过程中实现，而且还可以在 TCP 连接之后的数据传送过程中实现。假设主机 A 是服务器 B 的授权用户，攻击者主机 C 想冒充 A 获取相应授权服务器或者是想破坏 A、B 之间的通信，只要主机 C 能够猜测出要攻击的系统用于下一次 TCP 连接时使用的初始序列号，则 C 就能够欺骗服务器 B，通过假冒向该服务器发送 SYN/ACK（同步/确认）报文来欺骗服务器。其与主机 C 的 TCP 连接已经建立，这样，服务器认为与主机 A 的 TCP 连接实际上是与主机 C 的 TCP 连接。如果与此同时，主机 C 还利用其他攻击（如拒绝服务攻击）使主机 A 陷入瘫痪状态，主机 C 就可以在服务器 B 完全信任的情况下任意长时间进入服务器，并向服务器发送任意数据，而服务器 B 却认为这些数据是从它所信任的主机 A 发送过来的。

不同的系统在进行 TCP 连接时随机产生初始序列号的方式不同，总结起来，大致有以下几种类型。

1）传统模式：系统产生 ISN 的方式是每秒增加 128000，如果有连接出现，每次连接将把计数器的数值加 64000。

2）利用随机数生成函数产生：在 Linux2.0、OpenVMS 等操作系统中，系统利用随机数生成函数来产生一个 32 位的 ISN。在这种方式中，由于采用生成随机数的算法及生成随机数时所选的种子点都很难获知，因此，这种情况下攻击者实现其猜测的可能性很小，但是由于计算机中利用随机数函数产生的随机数都属于伪随机数，并且 ISN 还必须是在 32 位的范围内，因此前后序列号不可避免地会出现重复的情况，这在很大程度上降低了 TCP 的可靠性。

3）利用系统的时间间隔来产生。例如，在 Windows 98 系统中，ISN 就直接取两次建立连接在时间上相差的毫秒数。

4）常数：如在 3Com 的集线器和 Apple LaserWriter 打印机上都是使用一个固定的常数作为 ISN，前者为 0x803，后者为 0xC7001。

5）采用递增常数方式实现：如在一些打印机上每次都以一个固定的常数增量来以递增的方式产生下一次 TCP 连接的 ISN。

在整个欺骗过程中，对 TCP 序列号的猜测往往要进行很多次，这就可能会出现以下多种情况。

1）如果序列号猜测正确，主机 C 向主机 B 发送的数据就会被接收到缓冲区中。

2）如果序列号小于主机 B 所期望的序列号，主机 B 会认为是重复数据而将其丢失。

3）如果序列号大于主机 B 所期望的序列号并且小于 TCP 的接收窗口的当前接收范围，则该数据包将被暂时缓存到一个队列中等待处理，因为此时主机 B 认为后发送的数据先到达。

4）如果序列号不是目的主机所期望的序列号，并且还不在当前的 TCP 接收范围内，数据包将被丢弃。

利用 TCP 序列号猜测欺骗攻击可以达到很多攻击目的，常见的攻击目的如下。

1）发送虚假信息：此时攻击者让授权用户与服务器进行正常的连接和通信，在双方通信过程中的数据流中插入虚假的信息或修改某些信息，从而破坏通信信息的真实性和完整性。

2）拒绝服务攻击：攻击者一旦可以劫持合法授权用户，就可以干涉服务器与用户的通信，使得用户无法获得或无法正常连续地获得服务器所提供的服务，从而达到实现拒绝服务攻击。而这种拒绝服务的攻击并不通过像通常的 DoS（Denial-of-Service，DoS）攻击那样使服务器无法响应来实现，因此更具隐蔽性和欺诈性，使得管理员几乎察觉不到系统异常。

2．TCP SYN 洪泛攻击原理

TCP SYN 洪泛攻击是通过违反常规的执行 TCP3 次握手过程来实现的。通常情况下，建立一个 TCP 连接首先是用户向服务器发送一个 SYN 请求，服务器在收到 SYN 请求后会根据请求中的源 IP 地址向连接建立请求者发送 ACK 报文确认连接建立，通过这样的 3 次握手方式就正确无误地建立了一个 TCP 连接。TCP SYN 洪泛攻击通过消耗服务器的两类资源来达到攻击的目的。

（1）SYN Cache（请求队列）

在 TCP 连接的建立过程中，每当服务器接收到一个 TCP 连接建立请求，就向连接建立请求者回复 SYN+ACK 信息，同时将该请求插入到请求队列中等候处理。如果攻击者使用大量的并不存在的 IP 地址同时向服务器发送 TCP 连接建立请求，服务器就会响应大量的连接建立请求，并试图等待源 IP 的连接建立确认。如果这些 IP 地址根本不存在，则这些 IP 地址对于服务器来说是不可达的，那么，服务器就永远收不到任何回复确认信息，这个连接就处于半开状态一直待在请求队列中直到超时，而在这等待的时间里，系统资源得不到释放，由于请求队列的长度是有限的，大量半开状态的连接必将使请求队列溢出，使服务器无法处理新的连接建立请求，从而使服务器无法向正常的用户提供服务。

（2）SYN Cookie（缓冲区）

TCP 在将连接建立请求插入到请求队列的同时，会在本机内存中开辟相应的内存空间用于建立相关的数据结构和保存状态。大量的 SYN 请求信息必将使服务器开辟大量的缓冲区，从而将系统的内存资源消耗尽，使得服务器无法再进行其他工作，最终陷入瘫痪状态。

TCP SYN 洪泛攻击不仅仅可以达到瘫痪服务器的目的，同时，大量的 SYN 请求也会将网络带宽资源耗尽，使得网络系统无法正常工作。

目前 Internet 上占主导地位的 DDoS（分布式拒绝服务攻击）最主要的攻击方式都来自于 TCP SYN 洪泛攻击，所占比例为 90%～94%。当然，还可以利用其他的 TCP/IP 安全漏洞进行 DoS 和 DDoS 攻击。据统计，利用 UDP 的攻击报文占 2.4%～5%，利用 ICMP 的占 2.1%～2.6%，利用其他协议的占 2.06%～2.92%。

3．对应策略

基于 TCP 的攻击最终是利用 TCP 本身的工作机理实现的，要想根除这类攻击，最根本的方法是修改 TCP。针对序列号欺骗攻击，主要是修改 TCP 中 ISN 的生成算法，使得攻击者无法有效地猜测出序列号，而对于 TCP SYN 洪泛攻击，则需要建立相关的连接建立处理措施。例如，服务器并不能将所有的连接建立请求插入到请求队列中，同时，也改变在 TCP 连接建立请求之时就开辟相应的内存空间创建相应的数据结构的传统做法，而是等待连接完全建立过后才分配

其缓冲区空间。

而对系统、协议的更改并不是一朝一夕的事情。针对 TCP 攻击，目前可以采用的比较实际、有效的措施主要可归纳为以下几种。

（1）采用无状态 TCP 连接

当收到 TCP 连接建立请求时，并不按照通常的运行方式为其分配相应的数据结构和存储状态信息，而只是向源 IP 发送回复确认信息，将 TCP 连接建立保持在一种无状态的握手方式中，直到收到来自合法的 IP 连接建立确认时为其分配相应资源并存储相应状态。

（2）过滤防范措施

建立有效的 IP 地址信任关系，合理管理内部 IP 地址，建立合理的内部网拓扑结构，使得服务器可以将一些不信任的 TCP/IP 请求过滤，同时可以防范攻击者利用本地 IP 地址进行欺骗来猜测初始序列号，或者利用包过滤性防火墙来达到对不信任的 TCP/IP 连接请求的过滤目的。

（3）代理服务器型防火墙

利用代理服务器型防火墙（Proxy Firewall）来代理服务器进行 TCP 的二次握手，防火墙在处理 TCP 连接建立请求的同时还截取所有的半连接，这样就能够非常有效地保护内部服务器的安全。

（4）利用攻击检测技术

根据针对 TCP 的攻击的特征建立相应的检测手段和系统，如针对序列号欺骗采用对同一主机在短时间内所发送来的大量重复连接建立请求进行检测，对于 TCP SYN 洪泛攻击则可以建立简单的流量异常检测方式，并在检测出异常状况时及时向网络管理员提出预警。

（5）改进 ISN 算法

像 TCP 这样面向连接的协议使用了序列号机制，这种机制提供了一种确认方法。TCP 为每一个连接请求选择一个 ISN，为了防止因为延迟、重传等扰乱的 3 次握手，不能随便选取 ISN，不同系统采取不同算法或精心构造难以猜测的 ISN 算法。理解 TCP 如何分配 ISN 及 ISN 随时间变化的规律，对于理解 IP 欺骗攻击很重要。

（6）利用定时器

只有当建立连接后，才可以使连接建立定时器无效，也就是说，在不同开放连接建立过程中，当主机收到一个 ACK 时，定时器应置为无效，使状态转移到 ESTABLISHED。只有 CLOSED 等少数几种状态与定时器无关，入侵主机才可能会迫使 TCP 转移到这些状态，因为该状态不受任何定时器制约，如果入侵者不发送适当的包，主机可能会被阻塞在这个状态。

利用 TCP 序列号及 TCP 同步信息进行攻击是 TCP 本身的安全漏洞给网络带来的重大隐患和安全威胁，上文从 TCP 工作原理入手，深入剖析了基于 TCP 的安全攻击实现机理及实现途径，并提出了一定的应对策略。

3.3.12 TCP 的主要特点

1. 支持可靠性的面向连接服务

面向连接对可靠性的保证首先是它在进行实际数据传输前，必须在源进程与目的进程建立

一条传输连接。创建进程到进程的通信，TCP 使用端口号来完成这种通信。假如由于某种原因，传输连接建立不成功，则源进程不会像 UDP 一样向目的进程发送数据报。在发送方 TCP 和接收方 TCP 之间建立连接，将数据流分割成为可传输的单元，将它们编号，然后逐个发送它们。接收方 TCP 等待属于同一个进程的所有不同单元的到达，检查哪些单元没有发生差错，并将它们作为一个流交付给接收进程。当整个流发送完毕后，传输层关闭这个连接，同时面向连接传输的每一个报文都需要接收方确认，未确认报文被认为是出错报文。

2．支持流传输

流（Stream）是一个无报文丢失、重复和失序的正确的数据序列。流相当于一个管道，从一端放入什么，从另一端可以照原样取出什么。由于 TCP 同样也是建立在不可靠的网络 IP 之上，IP 不能提供任何可靠性机制，所以 TCP 的可靠性完全由自己实现。TCP 采用的最基本的可靠性技术是确认与超时重传，流量控制也是保证可靠性的一个重要措施。假如没有流量控制，可能因为接收缓冲区溢出而丢失大量数据，导致许多重传，TCP 采用可变窗口的方法进行流控制。此外 TCP 还要进行拥塞控制，以进一步提供可靠性，传输协议为实现可靠的流传输，付出了大量开销。

为了进行流报文交付，发送 TCP 和接收 TCP 都要使用缓存。发送 TCP 使用发送缓存来存储从发送应用程序传来的数据，发送应用程序交付数据的速率是它产生数据的速率。例如，如果用户是在键盘上键入数据，则逐一地交付给发送端 TCP；如果数据是从文件来的，则数据可能逐行或逐块地交付给发送 TCP，发送应用程序将数据写到发送 TCP 的缓存中。但是，发送不能为发送应用程序发出的每一个写操作创建一个数据报文段，TCP 可选择将几个写操作组合成一个报文段，以便使传输效率更高。接收 TCP 在收到报文段后就将它们存储在接收缓存中，接收应用程序使用读操作将数据从接收缓冲中读出，但是它不一定必须用一次操作将一个报文段中的所有数据全部读出，因为读的速率可以比接收速率低，因此在接收应用程序读完数据之前，数据仍留在缓存中。

3．支持全双工服务

TCP 支持数据可在同一时间双向流动的全双工服务，在两个应用程序已经建立传输连接之后，客户端与服务器进程可以同时发送和接收数据。TCP 连接可以从进程 A 向进程 B 发送数据，而在同一时间可以从进程 B 向进程 A 发送数据。当报文从进程 A 发往进程 B 时，它可以携带对进程 B 发来的报文的确认。当报文从进程 B 向进程 A 发送时，它也可以携带对进程 A 发来的报文的确认。确认随数据一起发送，当然如果一方没有数据可发送，就只能发送确认而没有数据，这种捎带确认的方法与数据链路层的帧确认方法类似。

4．支持可靠服务

TCP 是一种可靠的传输服务协议，它使用确认机制来检查数据是否安全和完整地到达。

图 3-28 给出了 TCP 与其他协议的层次关系，以及它与应用层协议的单向依赖关系。从图中可以看出，根据应用层协议与传输层协议的单向依赖关系，应用层协议可以分为 3 类：一类依赖于 UDP；一类依赖于 TCP；另一类既依赖于 UDP，又依赖于 TCP。依赖于 TCP 的应用层协议主要是需要大量传输交互式报文的一类应用，如虚拟终端协议 Telnet、电子邮件协议 SMTP、

文件传输协议 FTP 及超文本传输协议 HTTP 等。

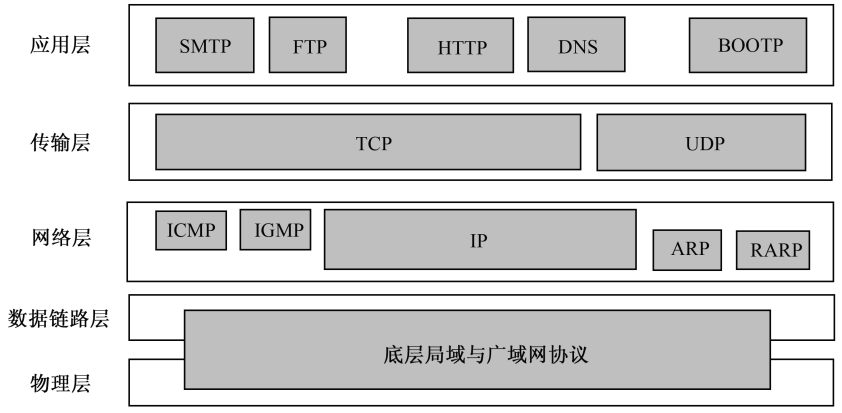


图 3-28　TCP 与其他协议的层次关系

小结

本章首先讨论了 UDP，它的正式规范是 RFC 768 [Postel 1980]，只包含 3 页内容。它向用户进程提供的服务位于 IP 层之上，包括端口号和可选的校验和。用 UDP 来检查校验和，并观察分片是如何进行的。接着，讨论了 ICMP 不可达差错，它是新的路径 MTU 发现功能中的一部分，当系统接收 IP 数据报的速率超过这些数据报被处理的速率时，系统可能发送 ICMP 源站抑制差错报文。使用 UDP 时很容易产生这样的 ICMP 差错。

TCP 提供了一种可靠的面向连接的字节流运输层服务，本章介绍和讨论了 TCP 首部中的各个字段。TCP 将用户数据打包构成报文段；它发送数据后启动一个定时器；另一端对收到的数据进行确认，对失序的数据重新排序，丢弃重复数据；TCP 提供端到端的流量控制，并计算和验证一个强制性的端到端校验和。许多流行的应用程序如 Telnet、Rlogin、FTP 和 SMTP 都使用 TCP。

参考文献

[1]　吴功宜．计算机网络高级教程[M]. 北京：清华大学出版社，2007.
[2]　郭银景，孙红雨，段锦．计算机网络[M]. 北京：北京大学出版社，2008.
[3]　孙红军．计算机网络[M]. 北京：机械工业出版社，2008.
[4]　张曾科，阳宪惠．计算机网络[M]. 北京：清华大学出版社，2006.
[5]　吴功宜．计算机网络. [M]. 2 版北京：清华大学出版社，2007.
[6]　高焕芝．新编计算机网络基础教程[M]. 北京：清华大学出版社，2008.

第 4 章 应用层

前面章节介绍了计算机网络的通信功能，本章主要介绍应用层进程如何使用这些通信功能为用户提供各种服务。应用层的主要内容是规定应用进程在通信时所遵守的协议。应用进程是指为了解决具体的应用问题而彼此通信的进程。本章的主要内容包括 InternetDNS、文件传输协议、电子邮件协议和万维网等著名的应用层协议及应用系统。

4.1 应用层简介

应用层位于网络体系结构的最高层，Internet 技术的发展极大地丰富了应用层的内容。每个应用层协议都是为了解决某一类应用问题，如文件传输、电子邮件、网站访问等而制定的。这类问题通常是通过位于不同主机上的多个应用进程之间的通信和协同工作来解决的，应用层的主要内容就是规定应用进程通信应该遵循的协议。

4.1.1 应用层协议及其与低层协议的关系

1．主要的应用层协议

在 TCP/IP 参考模型中，应用层是参考模型的最高层。应用层包括所有的高层协议，并且不断有新的协议加入。目前，应用层协议主要有以下几种。

1）InternetDNS：用于实现网络设备名称到 IP 地址映射的网络服务。

2）文件传送协议（File Transfer Protocol，FTP），用于实现 Internet 中交互文件传输功能。

3）简单邮件传送协议（Sample Mail Transfer Protocol，SMTP），用于实现 Internet 中电子邮件传送功能。

4）超文本传送协议（Hyper Text Transfer Protocol，HTTP），用于 WWW 服务。

5）简单文件网络管理协议（Simple Network Management Protocol，SNMP），用于管理与监

视网络设备。

2．应用层协议与低层协议的关系

按照层次结构思想，对计算机网络模块化的研究结果使 Internet 形成了一组从上到下单向依赖的协议簇。图 4-1 给出了 TCP/IP 参考模型应用层协议与低层协议之间的关系。

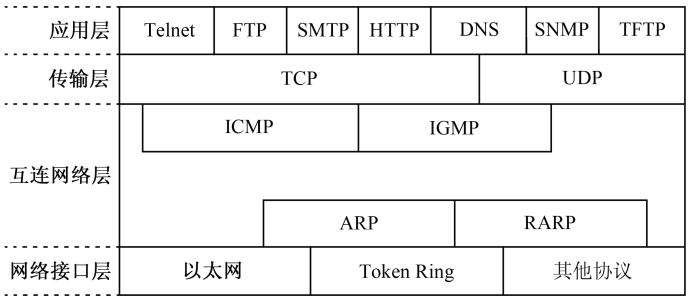


图 4-1　TCP/IP 参考模型应用层协议与低层协议之间的关系

应用层协议可以分为 3 种类型：一类依赖于面向连接的 TCP；一类依赖于无连接的 UDP；而另一类则既依赖于 TCP，又依赖于 UDP。其中，依赖 TCP 的主要有远程登录协议（Telnet）、简单邮件传送协议（SMTP）、文件传送协议（FTP）、超文本传送协议（HTTP）等；依赖 UDP 的主要有简单网络管理协议（SNMP）、普通文件传送协议（TFTP）等；既依赖 TCP 又依赖 UDP 的主要有 DNS 等。

ARP/RARP 并不属于单独的一层，它介于物理地址与 IP 地址间，起着屏蔽物理地址细节的作用。IP 可以建立在 ARP/RARP 上，也可以直接建立在网络硬件接口协议上。TCP、UDP 都要通过 IP 来发送和接收数据。

4.1.2　网络应用进程交互的 C/S 模式

在 Internet 应用层中，网络层进程以什么方式相互通信并协同工作呢？最重要的应用进程交互方式就是 C/S 模式。在 Web 环境下，C/S 模式又演进为基于 Web 的 C/S 模式，称为浏览器/服务器（Browser/Server，B/S）模式。Internet 中的很多网络应用（如 FTP、TFTP、DNS、E-mail、WWW 和 Socket 提供的网络通信机制）采用 C/S 模式。除了 C/S 和 B/S 模式外，还出现了一种新的进程通信模式 P2P（Peer to Peer）。P2P 技术现在已受到人们的普遍关注并成为研究热点。

1．C/S 模式的基本概念

C/S 模式也可称为 C/S 计算模式或 C/S 模型。

在计算机网络中，每台连网的计算机既要为本地用户提供服务，也要为网络中其他主机的用户提供服务。因此，每台连网的计算机的硬件、软件与数据资源既是本地用户可以使用的资源，又应该是网络上的其他主机用户可以共享的资源。网络的每一项服务都对应一个“服务程序”进程。这些进程要为每一个获准的网络用户请求执行一组规定的动作，以满足用户网络资

源共享的需要。

进程通信的实质是进程之间的相互作用。网络环境中的进程通信要解决的一个重要问题是确定进程间的相互作用模式。在 TCP/IP 体系中，进程间的相互作用主要采用 C/S 模式。

在 C/S 模式中，客户与服务器分别表示相互通信的两个应用程序的进程。客户向服务器发出服务请求，服务器响应客户的请求，提供客户所需要的网络服务。发起本次进程通信、请求服务的本地计算机的进程称为客户进程，远程计算机提供服务的进程称为服务器进程。图 4-2 是 C/S 模式的进程相互作用的示意图。

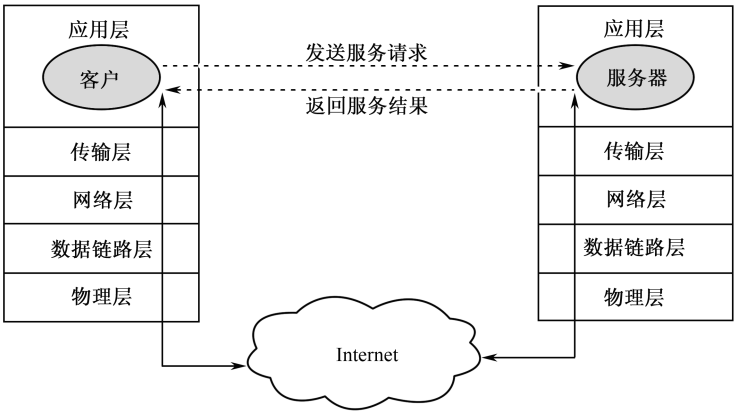


图 4-2　C/S 模式的进程相互作用

2．采用 C/S 模式的主要原因

TCP/IP 采用 C/S 模式作为应用程序间相互作用的最主要形式，主要原因如下。

1）从技术方面讲，互联网上不同主机进程之间进行通信，其重要特点是主机发起通信完全是随机的，一台主机上的进程不知道另一台主机的进程会在什么时候发起一次通信。因此需要一种机制，能够适应这种随机性。C/S 模式很好地解决了上述技术问题，每次通信过程都由客户进程主动发起，而且是随机的，服务器进程从开机起就处于等待状态，随时准备对客户的请求做出及时响应。C/S 模式为通信进程建立了联系，为它们之间的数据交换提供同步，客户主动发起请求，服务器等待，被动响应。

2）从实际应用方面讲，C/S 模式的重要特点是非对等性相互作用，即客户请求服务，服务器提供服务。一般提供服务的计算机要比请求服务的计算机拥有更好、更多的软硬件资源和更强的处理能力。C/S 模式很好地适应了 Internet 上资源分布不均的客观现实，充分地利用了网络资源。

3）C/S 模式优化了网络计算，提高了网络的利用率。客户可以请求服务器进行如数据库查询等类的大型计算，客户接收用户（User）的查询请求，形成查询报文传给服务器（这里是数据库服务器），服务器执行大型数据库的查询，之后将查询结果传回给客户，客户进行结果显示，提供友好的人机界面。因此，客户和服务器分工合作，协同完成计算，网络上传输的只是简短的查询请求和结果。

3．B/S 模式

WWW 产生后，Web 环境下的应用也广泛采用 C/S 模式。在 Web 的 C/S 模式中，客户是浏览器，万维网文档所驻留的计算机运行服务器程序，即万维网服务器（Web Server）。客户向服务器发出信息浏览请求，服务器向客户送回客户所要的万维网文档，以页面（Page）的形式显示在客户的屏幕上。万维网的这种基于 Web 的 C/S 模式称为 B/S 模式，也可称 B/S 计算模式。

B/S 模式也属于 C/S 模式，但它有自己的特点。B/S 模式的一个重要特点是平台无关性，主流语言 Java 和 HTML 等都可以做到与软硬件平台无关。而在一般的 C/S 模式应用中，不同的网络操作系统下可能使用不同的语言和开发工具。B/S 模式的客户端变“瘦”，其功能只是一个多媒体浏览器和 Java 虚拟机，而一般的 C/S 模式是“胖”客户端，客户端除了负责事件输入和图形显示外，还可能要进行应用逻辑和业务处理，它是开发的重点。B/S 模式可以提供多层次连接，常常是浏览器/Web 服务器/应用服务器的形式，广泛使用 Browser/Web Server/DBMS 这 3 层连接，Web Server 和数据库管理系统（DBMS）连接，并可读取数据库中不断更新的数据，这样浏览器就可在网页中浏览动态的数据。

4．讨论 C/S 模式时需要注意的几个问题

1）使用计算机的人是“用户（User）”而不是“客户（Client）”。C/S 模式中的客户和服务器都指的是进程，即计算机软件。过去曾将 Client 译为“客户机”，但这不是标准译名，而且使用“客户机”这种译名容易使人误认为 Client 是硬件。需要指出的是，由于运行服务器进程的机器往往有许多特殊的要求（不同于普通的 PC），因此人们经常将主要运行服务器进程的机器（硬件）不严格地称为服务器。例如，“这台机器是服务器”，其实这句话的意思是“这台机器（硬件）主要用来运行服务器进程（软件）”。于是，服务器一词有时指的是软件，但有时指的是硬件。在本书中，在涉及允许客户或服务器进程的机器时，我们常说“客户端”和“服务器端”。

2）客户和服务器的角色有时可以互换。例如，有两台计算机 A 和 B，其中主机 A 的进程 PA 向另一台主机 B 的进程 PB 提出连接请求，希望主机 B 向主机 A 提供服务，那么在这次通信中，进程 PA 是客户进程，而进程 PB 是服务器进程。反之，则进程 PB 是客户进程，而进程 PA 就是服务器进程。

4.1.3 进程通信中 C/S 模式的实现方法

（1）服务器处理客户并发请求的基本方法

C/S 模式是采用“请求驱动”方式工作的。为了实现服务器的功能，在服务器的设计中要解决服务器的并发请求处理能力、并发服务器的进程标识及服务器安全等主要问题。

在网络环境中，客户进程发出请求完全是随机的。在同一个时刻，可能有多个客户进程向一个服务器发出服务请求。因此，服务器必须有处理并发请求的能力。服务器处理并发请求的方案有两种：一是采用并发服务器（Concurrent Server）；二是采用重复服务器（Iterative Server）。

（2）并发服务器的基本工作原理

并发服务器的核心是一个守护（Daemon）进程，它随系统的启动而运行。Daemon 又写为 Demon，它是 UNIX 中的一个术语，是一种程序或进程，处于后台工作，在某种条件满足时被激活，并开始进行处理。

在没有客户的服务请求达到时，并发服务器处于等待状态。一旦客户的服务器请求到达，服务器根据客户的服务请求的进程号，激活相应的子进程，由子进程为客户提供服务，而服务器继续回到等待状态。并发服务器称为主服务器（Master），子进程又称从服务器（Slave）。并发服务器利用主/从服务器结构来达到处理并发请求的目的。因此，服务器必须拥有一个全网熟知的进程地址。网络中的客户进程可以根据服务器进程的熟知地址，向服务器提出服务请求。在实现进程通信的过程中，客户与服务器进程分别形成自己的半相关三元组，即传输层协议、IP 地址、端口号，然后客户根据服务器进程的熟知地址建立全相关的五元组。并发服务器建立连接的工作过程如图 4-3 所示。

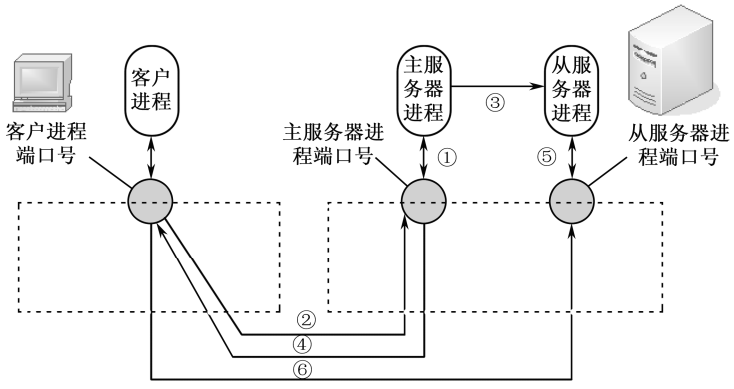


图 4-3　客户/并发服务器传输连接的建立连接

注：

① 主服务器在熟知的端口号上准备接收客户的连接建立请求。

② 客户向服务器发出连接建立请求。

③ 主服务器在接收到客户端的连接建立请求后，激活响应的从服务器。

④ 主服务器通知客户从服务器的端口号，并关闭与客户的连接。

⑤ 从服务器准备接收客户的连接建立请求。

⑥ 客户向服务器发送请求连接建立请求。

（3）重复服务器的工作原理

重复服务器不产生从服务器进程。重复服务器包含一个请求队列，客户请求到达后，首先进入队列之中，服务器按先进先出（First In First Out，FIFO）原则对这些请求逐一做出响应。

（4）重复服务器与并发服务器的比较

并发服务器的一个优点是从服务器独立响应请求而不依赖于主服务器，不同从服务器可以按不同的方式响应请求，具有很大的灵活性。因为多个从服务器在并发地响应不同的请求，所以并发服务器的另一优点是实时性好。但并发服务器因产生了多个子进程而增加了系统开销。

重复服务器的优点是可以有效地控制请求处理的时间，缺点是重复服务器处理客户的服务请求数量受到队列长度的限制。

因此，并发服务器适用于面向连接的服务类型，而重复服务器适用于无连接的服务类型。Socket 提供的网络通信机制使用 C/S 模式，程序员可以控制使用并发服务器或重复服务器。

4.2 InternetDNS 简介

4.2.1 DNS 原理

Internet 对计算机的命名方案称为 DNS。语法上，计算机的域名由一系列字母和数字构成的段组成。例如，北京信息科技大学光电信息与通信工程学院的域名为 gd.bistu.edu.cn，cn 代表中国，edu 代表教育部，bistu 代表北京信息科技大学，gd 代表光电信息与通信工程学院。由此看出域名是有层次的，域名中最右边的部分都表示国家，最左边的部分代表该台计算机的名称。

域名和 IP 地址是一一对应的，域名易于记忆，用得更普遍。当用户要和 Internet 上某台计算机交换信息时，只需使用域名，网络会自动转换成 IP 地址，找到该台计算机。

DNS 规定了最高域的值，称为 DNS 的顶层。当一个组织希望参加 DNS 时，必须申请一个顶层域下的一个域名。一旦一个组织拥有一个已申请的域，就可以决定是否设置进一步的层次规模。一般来说，对于一个规模小的组织可以不再设置下一次的域，而对于规模大的组织就有必要设置多层结构。

因为域名是一个逻辑概念，所以它不必与物理地点相一致。而且对一个组织来说有不同的部门，可以根据部门的实际情况选择不同层次的域名构造。

命名系统的一个特点是自治，即允许每个组织为计算机设置域名或改变这些域名。大多数具有 Internet 连接的组织运行一个域名服务器，称 DNS 服务器。每当应用需要将域名翻译为 IP 地址时，应用成为 DNS 的一个客户。这个客户将待翻译的域名放在一个 DNS 请求信息中，并将这个请求发给 DNS 服务器。服务器从请求中取出域名，将其翻译为对应的 IP 地址，然后，在一个回答信息中将结果地址返回给应用。

4.2.2 域名的分级

早期的 Internet 使用非等级的名称空间来描述域名。整个 Internet 上的计算机使用名称的集合组成一个非等级的名称空间，每个名称由字符序列组成，由中心网点[即网络信息中心（Network Information Center，NIC）]来管理名称空间。

非等级名称空间的主要优点是名称简单和短小。然而，主要缺点是非等级名称空间由于技术和管理上的原因，不能推广至具有大量计算机的网络。由于名称取自单一的标识字符集，随着网点数的增加，潜在冲突也随之增多；名称管理机构的管理工作负担也随着网络规模的扩大而加重。

Internet 不采用集中命名的机制，而采用分布的名称空间管理机构，使名称和地址的映射任务分布执行。

名称空间的划分机制既要高效地支持名称映射，又要保证名称分配能自行控制。如果只考虑映射的效率，只需采用非等级名称空间，通过在多个映射机器之间划分名称来减少通信流量；如果只考虑管理的方便，只需使管理机构的授权容易，但会增加名称映射的开销和复杂性。

Internet 采用分级名称空间的管理，如同一个大单位的管理，在最高层划分名称空间，并制定代理负责下一级的名称管理。例如，一个名称空间的表示为 local.site，site 是由中心管理机构授权的网点名，local 是受 site 网点控制的名称的一部分，点是分隔它们的分界符。还可以进一步细分。例如，增加一个 group，名称空间表示为 local.group.site。原则上，可继续细分到足够小以便管理。

TCP/IP 互联网的分组命名方案不一定要根据物理位置来划分，可以按照组织结构或部门类型来划分。也就是说，TCP/IP 命名方案允许任意选取分级名称空间的管理机构，而与物理连接无关。

4.2.3　InternetDNS

DNS 有两个概念上独立的要点：一个抽象的，即指明名称语法和名称的授权管理规则；另一个是具体的，即指明一个分布式计算系统的实现，它能高效地将名称映射到地址。

DNS 将域名的每一部分称为标识符。例如，域名 gd.bistu.edu 含有 3 个标识符。在域名中一个标识符的任一后缀又称域。上面所述中最低层的域是 gd.bistu.edu，表示北京信息科技大学光电信息与通信工程学院的域名；第二级域是 bistu.edu，表示北京信息科技大学的域名；最高层域是 edu，表示教育部门的域。

理论上，可以用任意标识符规则定义一个抽象分级的名称空间。然而，大多数用户还是沿用 InternetDNS 正式使用的分级标识符规则。因为 InternetDNS 命名方案能适应于多种组织结构，允许它们选择地理的或组织的命名分级，而且不需改动命名，就可将该组织的 TCP/IP 网方便地连接到整个 Internet 上。

在概念上，顶层的名称可采用两种完全不同的命名分级，即地理的和组织的。前者按照国家或地区来划分，后者按组织的类型划分。

Internet 的域名空间分成 3 类，即通用域、国家域和反向域，如图 4-4 所示。

通用域按照通用特性定义已注册的计算机。在图 4-4 所示的树结构中的每个节点定义一个域，作为域名空间数据库的索引。

由图 4-5 可知，基本的通用域的第一级用 3 个字符的标识符表示，共 7 种，如表 4-1 所示。

国家域的格式类型同于通用域，但第一级只用两个字符的标识符表示，如图 4-6 所示。

反向域用来将一个地址映射到名称。反向域的应用实例如下：当域名服务器从客户机 接收到一个请求，服务器有一个包含授权客户的文件，但它只列出客户的 IP 地址。为了确定该客户是否在授权列表中，可以送一个查询请求到 DNS 服务器，并将地址映射至名称。反向域的机制如图 4-7 所示。

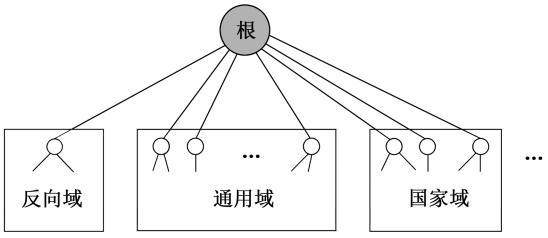


图 4-4　Internet 的 DNS

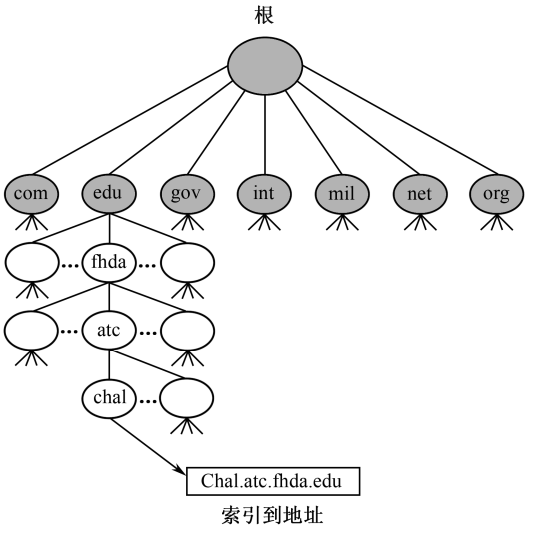


图 4-5　通用域

表 4-1　通用域标识符

标识符	描 述	标识符	描述
com	商业组织	mil	军事部门
edu	教育、研究机构	net	网络支持中心
gov	政府部门	org	非营利组织
int	国际组织		

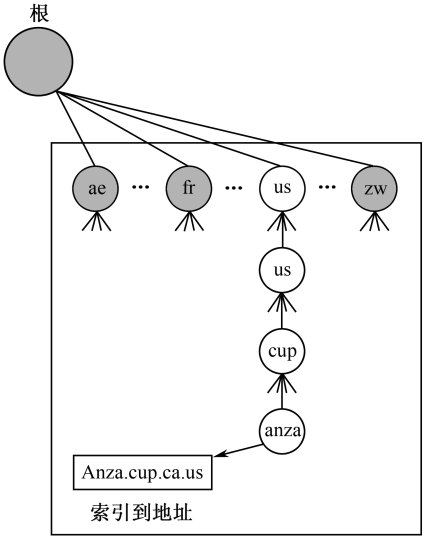


图 4-6　国家域

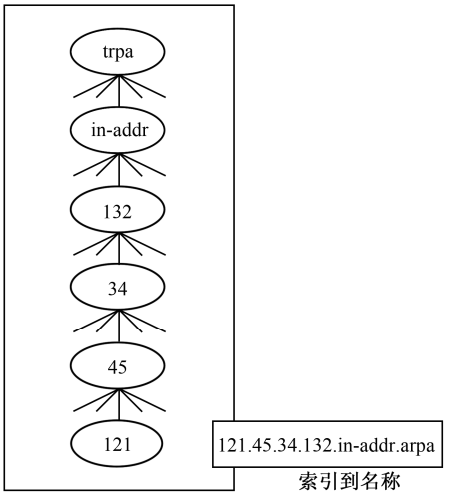


图 4-7　反向域

4.2.4　域名和地址映射

域名方案应包含一个高效、可靠、通用的分布式系统，实现名称对地址的映射。系统是分布的，由分布在多个网点的一组服务器协同操作解决映射问题；系统是高效的，大多数名称映射在本地操作，只有少数名称的映射需要在互联网上通信；系统是通用的，因为它不限于仅使用机器名；系统是可靠的，单台计算机故障不会影响系统的正确运行。

名称对地址的映射由一组名称服务器完成，名字服务器是提供名字对地址转换，域名对 IP 地址映射的服务器程序。

图 4-8 是一个树结构的域名服务器的概念布局。树的根是识别顶层域的服务器，并知道解析每个域的服务器。给定要解析的名字后，根可为该名字选择一个正确的服务器，并逐级往下搜索，最后将结果返回。

概念树中的连接不表示物理网的连接，而是解析域名的一种逻辑连接。服务器树是用于 Internet 通信的一种抽象结构。

从概念上讲，域名转换自上而下进行，从根服务器开始，逐级处理，直到树叶上的服务器。客户机必须知道如何与名字服务器联系，而域名服务器必须知道根服务器的地址。域名服务器使用协议端口通信，以便客户机方便地与域名服务器通信。

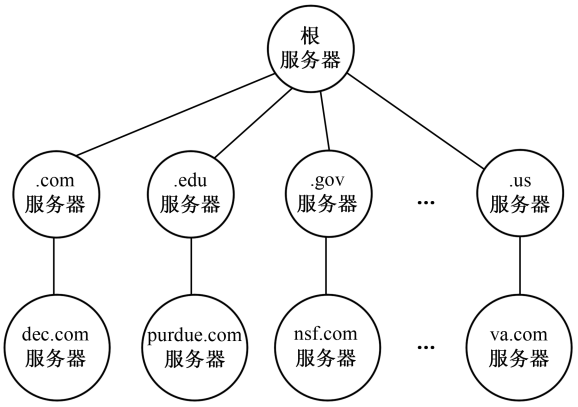


图 4-8　树结构域名服务器的概念布局

4.3　文件传输协议

4.3.1　文件传输协议

文件传输协议（File Transfer Protocol，FTP）是由 TCP/IP 提供的标准机制，可将文件从一个主机传送到另一个主机。这是网络和互连环境下常用的环境。

FTP 的实现需要解决以下问题：两个系统可能使用不同的文件名字约定，两个系统可能用不同的方法表示文本和数据，两个系统可能有不同的目录结构。

FTP 和其他 C/S 应用的区别在于在两个主机之间建立两个连接。一个连接应用于传送数据，而另一个连接用于命令和响应等控制信息。将两者分开，可使 FTP 的运行效率更高。控制连接使用十分简单的通信规则，它只需传送一行命令或一行响应。而数据连接则需较为复杂的规则，因为传送的数据有不同类型。

FTP 模型如图 4-9 所示。客户机有 3 个组成部分：用户接口、控制进程和数据传送进程。

服务器有两个组成部分：控制进程和数据传送进程。

在整个 FTP 交互会话过程中，控制连接一直保持。而数据连接在每个文件传送时打开和关闭。两个 FTP 连接使用不同的策略号和端口号。

Internet 上实现资源共享的最基本手段之一就是匿名 FTP（Anonymous FTP）。对 Internet 来说，也是极为有利的。利用 FTP 协议，可将成千上万兆位的数据文件从一个地方迅速传输到另一地方，FTP 服务器（ftpd）通过 FTP 协议与 FTP 客户程序之间进行信息交换。一般地，将数据从 FTP 客户程序传输到 FTP 服务器，称为“数据上载”，反之，FTP 客户程序从 FTP 服务器获取数据，称为“下载数据”。

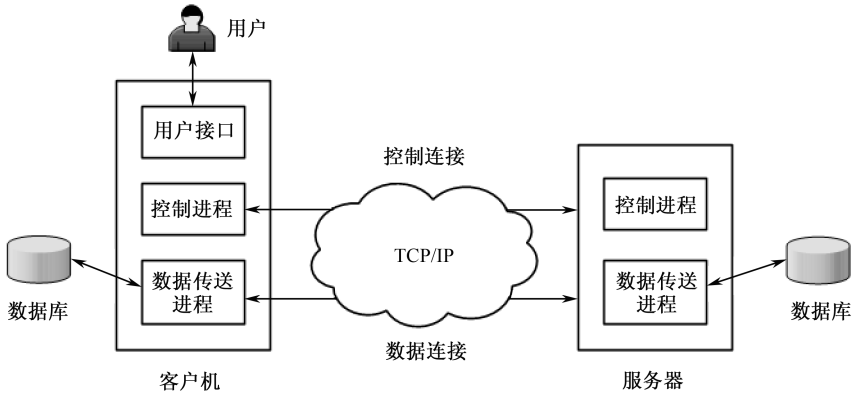


图 4-9　FTP 模型

FTP 服务器可提供两种访问形式：内部用户 FTP 和匿名 FTP。匿名 FTP 是 Internet 的公共信息服务，访问范围限于匿名 FTP 区域——FTP 服务器定义的字文件系统。内部用户 FTP，适用于在主机上有账号的用户，用户在输入正确的账号和口令字后，可以访问整个文件系统中有读权限的文档，并可以任意上载数据到有写权限的目录。这种方式隐藏着不安全的因素，因为用户的账号和密码在网络上明文传输，可能在中间环节被人窃取，并用于 Telnet 等远程登录，执行任何破坏性的指令。

匿名 FTP 服务是最常用的方式，用户只需以“Anonymous”登录，输入自己的电子邮件地址作为口令字，即可访问并下载所提供的信息资源。有时，还可以将自己认为有用的资源上载到 FTP 服务器。FTP 服务器并不严格地检查用户输入的口令字中的电子邮件地址，但在访问日志中记录了客户程序及其访问的细节。

建立 FTP 服务并不是一件轻松的事情。可以遵照 Internet 一些事实上的规则，来提供高质量的 FTP 服务，用户可以很方便地找到自己需要的文档资料。

下面简单介绍 FTP 系统的组成。

（1）FTP 服务器

FTP 系统服务实际上包含两个部分：服务器，即响应客户请求及传送文档；文件系统，即服务器文档扫描调用的区域。

FTP 服务器命名通常是 ftpd 或 in.ftpd。大多数 UNIX 系统都将 ftpd 和 ftp 作为标准集成在

操作系统中，一般在/usr/etc 目录。

（2）FTP 服务器的运行方式

FTP 服务器通常在系统超级服务 INETD 进程下运行。INETD 监听各个常用服务端口，如 FTP 服务控制 TCP 端口 21，当 FTP 客户程序访问一个服务器时，首先发送一个 TCP 包到目标主机的 21 端口，INETD 接收到这个 TCP 包后，从所请求的端口号确定启动 FTP 服务，生成子进程并执行 ftpd 的一个副本。如果同时有其他客户的 FTP 请求，INETD 同样地分配给一个 ftpd 来处理。

FTP 服务运行配置，在 INETD 的配置文件/etc/inetd.conf 中添加相应的一行设置，大多数 UNIX 系统的/etc/inetd.conf 中已有默认的 ftpd 定义。具体格式为“ftp stream tcp nowait root/etc/ftpd ftpd-1”。其中，“ftp”是协议域，告诉 INETD 对应此行的服务，INETD 通过对照/etc/services 找出 ftp 对应的服务端口号 21，根据监听到的请求端口号，确定启动何种服务。“stream”和“tcp”描述服务器接收的通信类型，此例中指两台主机之间的数据流通信，并基于 TCP 连接进行。FTP 服务使用 TCP 连接，其他有些服务基于 UDP 数据报协议通信。“nowait”通知 INETD 每个客户程序一次启动一个服务器进程；如果是“wait”，表示只能有一个服务器进程运行。“/etc/ftpd”和“ftpd -1”定义了 ftpd 执行程序的路径和服务器启动时的命令行参数。

在每次更新/etc/inetd.conf 配置文件后，需要使用“kill-HUP INETD 进程号”重启动 INETD。

（3）FTP 用户

在 INETD 下配置好 FTP 服务器后，需要在主机的/etc/passwd 中设置用户 ftp，因为 FTP 服务器在允许用户匿名访问 ftp 服务之前，首先检查 ftp 用户是否存在，如果不存在，ftpd 拒绝匿名用户访问。

ftp 用户设有唯一的用户号和组号，并且通过设置 ftp 用户的口令字为“*”及登录 shell 环境为不存在的/bin/false，来禁止 Telnet/rlogin 远程登录。另外，ftp 用户的登录目录也是很重要的，匿名用户访问的范围仅限于此目录 FTP 系统服务资源目录或称匿名 FTP 区域。例如：

```
ftp: * : 500 : 25 : Anonymous FTP User ： /home/ftp ： /bin/false
```

这种访问限制是 FTP 通过 chroot（）系统调用来实现的。

（4）FTP 服务命名

出于 Internet 习惯，FTP 服务常常通过在 DNS 系统中设立 CNAME 别名 ftp.domain.name 来命名。例如，CERNET 的匿名 FTP 服务是 ftp.edu.cn，在 DNS 数据库中有如下定义：ftp IN CNAME galaxy.net.edu.cn。这样，用户只要知道某个单位域名，就可以猜出此单位匿名 FTP 服务的名字。

4.3.2　镜像系统

文件服务器镜像系统（Mirror Sites）完成对远程匿名 FTP 服务器资源的本地镜像。在镜像描述文件中指定远程 FTP 服务器地址、登录名（Anonymous）及口令（E-mail address）、需要镜像的远程 FTP 服务器的目录或文件、本地 FTP 服务器上的文件存放路径和权限控制码，镜像系统就能够根据镜像描述文件，使用 FTP 协议自动登录到远程 FTP 服务器，进入相应的目录，取

得该目录下的文件列表，与本地目录下的文件列表进行比较。如果远程 FTP 服务器的文件大小或修改时间有所更新，或本地没有该文件，则将该文件取回。镜像系统定时定期进行以上工作，可以通过 UNIX Cron 设置命令参数来控制镜像系统定期检查和某次 FTP 失败后重试的时间。

目前流行的镜像系统软件是 Mirror-2.3 软件包，它是用 Perl 语言编写的程序，按照 FTP 协议，在运行它的主机与远程主机之间，按目录和文件结构进行数据传输。Mirror-2.3 是专为档案管理和通过 FTP 传输大量文件的需要而编写的软件包。

用户可以根据需要创建镜像点描述文件，并设置各种参数，如 package 软件包代号、site 镜像点地址（主机名或 IP 地址）、remote-dir 镜像目录、local-dir 本地存储目录、remote-user 远程 ftp 用户名、remote-password 远程 ftp 用户口令、get_patt 获取文件匹配、exclude_patt 忽略文件匹配、mail-to 电子邮件地址（可以将镜像日志通知管理员）等。

启动镜像服务，使用命令行“mirror –p 软件包代号 –d 镜像配置文件>镜像日志”。

4.4 远程登录协议

1．远程登录简介

Telnet 是 Internet 最早提供的服务，是 Tele-communication Network Protocol 的英文缩写，意指 Remote Login（远程登录）。它是 Internet 中用来进行远程访问的重要工具之一。远程登录功能允许用户与异地计算机进行动态交互，即用自己的键盘、鼠标等输入设备操作异地计算机，运行异地计算机上的软件，在自己的显示器上了解运行情况，看到运行结果。

在互联网上，大量运行分时系统的大型计算机有很高的运行速度、极大的存储容量、丰富的软件和庞大的数据资源。使用远程登录功能，用户可以在异地登录这些大型计算机，利用它的强大功能来完成自己在 PC 上难以完成的任务。

为了达到这个目的，人们开发了远程终端协议，即 Telnet 协议。它精确地定义了远程登录客户机与远程登录服务器之间的交互过程。通过 Telnet 协议，一台计算机可以作为远程主机的一个虚拟终端，通过网络远程利用服务器所提供的软件、硬件等资源完成自己的任务。

2．远程登录服务的主要作用

1）允许用户与远程计算机上运行的程序进行交互。

2）当用户登录到远程计算机时，可以执行远程计算机的任何应用程序，并且能屏蔽不同型号计算机之间的差异。

3）用户可以利用个人计算机完成许多只有大型计算机才能完成的任务。

3．Telnet 协议

系统的差异性通常是指不同厂家生产的计算机在硬件或软件方面的不同。系统的差异性给计算机系统的互操作性带来了很大困难，而 Telnet 协议可以解决多种不同计算机系统之间的互操作性问题。

不同计算机系统的差异性首先表现在不同系统对终端键盘输入命令的解释上。为了解决系统的差异性，Telnet 协议引入了网络虚拟终端（Network Virtual Terminal，NVT）的概念，它提供了一种专门的键盘定义，用来屏蔽不同计算机系统对键盘输入的差异性。

NVT 是一种标准格式。在客户端，客户软件在 TCP 连接传输之前把本地格式转变为 NVT 标准格式。在服务器端，服务器软件再把 NVT 格式转换为远程系统能够识别的格式。这样，有关键盘输入表示的差异性(不同操作系统对键盘的输入存在不同的表示方法)便被 NVT 屏蔽(来自于 IP 对底层网络的屏蔽)。这样才可以使得在运行 Windows XP 的 PC 上可以访问 UNIX 操作系统的远程主机。NVT 的原理如图 4-10 所示。

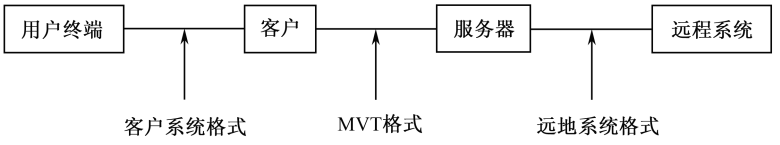


图 4-10　Telnet-NVT 原理图

4．Telnet 通信过程

Telnet 同样也采用 C/S 模式。在远程登录过程中，用户的实终端采用用户终端的格式与本地 Telnet 客户机程序通信。远程主机采用远程系统的格式与远程 Telnet 服务器进程通信。通过 TCP 连接，Telnet 客户机程序与 Telnet 服务器程序之间采用了网络虚拟终端 NVT 标准来进行通信。Telnet 客户机通信过程如下。

1）建立与远程主机的 TCP 连接。它在一个常用的 23 号 TCP 端口上使用一个套接字。如果远程主机上的 Telnet 服务器软件一直在这个常用的端口上侦听连接请求，则这个连接便将建立起来。

2）以终端方式为用户提供人机界面，从键盘上接收用户输入的字符。

3)把用户输入的字符串变成标准格式,并将用户的信息通过 Telnet 协议传送给远程服务器。

4）接收远程主机发送来的信息，并经过适当地转换显示在用户计算机的屏幕上。

5）把该信息显示在用户的屏幕上。

远程主机必须运行 Telnet 服务器软件，这样才能提供 Telnet 远程登录服务。Telnet 服务器软件将完成下列功能。

1）通知网络系统做好提供远程连接服务的准备。

2）不断地在常用的 23 号 TCP 端口上侦听用户的连接请求。

3）处理用户的请求。

4）将处理的结果通过 Telnet 协议返回给客户程序。

5）继续侦听用户的请求。

5．Telnet 的使用

Telnet 的使用操作比较简单。首先，运行 Telnet 客户程序，在运行 Telnet 程序时，可以直接在 Telnet 后缀一个远程主机的域名或 IP 地址来建立与该远程主机的连接（>Telnet 域名/IP 地

址），或者进入 Telnet 后用 open 命令来建立连接（Telnet>open 域名/IP 地址）。当连接建立起来后，系统会提示用户进行登录，即在“Login：”后面输入用户的注册名。除非是一个公共账户，否则系统提示用户输入口令。当用户的注册名和口令被远程主机确认后，用户计算机就成为该远程主机的一个终端，即可进行联机操作。

例如，假设要连接名字为 computer.xaut.edu.cn 的计算机，则输入“telnet computer.xaut.edu.cn”。

正如在前面所介绍的，所有 Internet 主机都有一个 IP 地址，也可以使用远程主机的 IP 地址进行登录：“telnet 202.200,112.9”。

运行 Telnet 程序后，它将开始连接所指定的远程机。当 Telnet 正在等待响应时，屏幕将显示“Trying…”（或类似的信息）。一旦连接确定（若主机距离远，可能会等待一段时间），将读到信息“Connected to test.xaut.edu.cn”。

假如有时 Telnet 不能确定连接，将会出现提示主机找不到的信息。例如，假如错误地输入“telnet test.xaut.com.cn”，将会出现“test.xaut.com: unknown host”。此时可以另指向一个主机名，或者中止执行该程序。常用的 Telnet 命令如表 4-2 所示。

表 4-2　常用的 Telnet 命令

常用命令	功能描述	常用命令	功能描述
help	联机帮助命令（特殊的 Telnet 命令）	status	显示状态信息
open	建立与远程主机的连接	toggle	改变工作参数（toggle?可显示详细参数）
close	正常结束远程会话，返回命令方式	quit	关闭 Telnet 会话，退出 Telnet 程序
display	显示工作参数	Z	退出 Telnet，进入暂停状态
mode	进入行命令或字符方式	<CR>	退出命令方式，返回 Telnet 的会话方式
send	向远程主机发送特别字符	?	与 help 命令等同，显示联机帮助信息
set	设置工作参数		

有许多因素都可能导致 Telnet 不能远程连接。3 个最常见的因素为计算机地址拼写错误、远程计算机暂时不能使用、所指向的计算机不再 Internet 上。

Telnet 一旦确定连接，即可同远程机对话。此时，许多主机会显示一些信息，一般用来确定计算机。一旦被接收，将会出现标准的提示符。例如，如果与一台 UNIX 远程机连接成功，将会出现“Login：”，输入用户名（账号）并按 Enter 键，将出现“Password：”，然后输入口令并按 Enter 键就可以进入系统。在远程主机的工作完成后，需要按常规方式“退出”，此时连接中断，Telnet 自动停止运行。

4.5　电子邮件协议

4.5.1　电子邮件简介

电子邮件是 Internet 上使用最多和最受用户欢迎的一种应用。电子邮件将邮件发送到收件

人的电子邮箱（Mail Box）中，收件人可随时进行读取。电子邮件不仅使用方便，而且还具有传递迅速和费用低廉的优点。现在电子邮件不仅可传送文字信息，而且还可附上声音和图像。

最初的电子邮件系统功能简单，缺乏内部结构格式的标准，计算机很难对电子邮件进行处理，用户接口也不好。但是经过人们的努力，于 1982 年制定出 APPANET 上的电子邮件标准。简单邮件传送协议和 Internet 文本报文格式已成为互联网事实上的标准。1984 年，CCITT 制定了报文处理系统（Message Handing System，MHS），即 X.400 建议书。1988 年，CCITT 修改了 X.400 建议书，它后来成为一个功能很强的电子邮件标准。

由于 Internet 的 SMTP 只能传送可打印的 ASCII 码电子邮件，因此在 1993 年又制定了新的电子邮件标准，即通用互联网邮件扩充（Multipurpose Internet Mail Extensions，MIME）标准。MIME 在其电子邮件首部中说明了电子邮件的数据类型（如文本、声音、图像、影像等）。在 MIME 电子邮件中可同时传送多种类型的数据。

4.5.2　电子邮箱和地址

一个电子邮件系统有 3 个主要组成部分：用户代理、电子邮件服务器和传输电子邮件所使用的协议，如图 4-11 所示。用户代理（User Agent，UA）是用户与电子邮件系统的接口，通常是在用户 PC 中运行的程序，其功能是给用户提供一个友好界面，来发送和接收电子邮件。

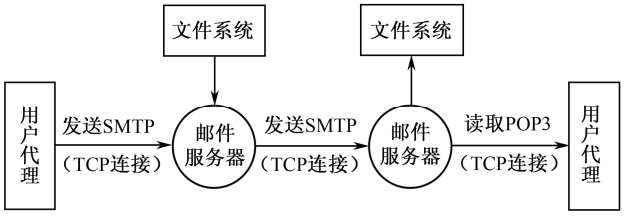


图 4-11　电子邮件系统的组成

使用电子邮件的前提是拥有一个电子邮箱。电子邮箱是由电子邮件服务的机构（一般是互联网服务提供商 ISP）为用户建立的。当用户向 ISP 申请 Internet 账户时，ISP 就会在它的 E-mail 服务器上建立该用户的 E-mail 账户。建立电子邮箱，实际上是在 ISP 的 E-mail 服务器磁盘上为用户开辟一块专用的存储空间，用来存放该用户的电子邮件。这样用户就拥有了自己的电子邮箱。用户的 E-mail 账户包括用户名（User Name）与用户口令（Password）。通过 E-mail 账户，用户即可发送和接收电子邮件。任何人可以将电子邮件发送到属于某位用户的电子邮箱，但只有电子邮箱的主人使用正确的用户名与用户口令，才可以查看电子邮箱的信件内容，或对其中的电子邮件做必要的处理。完整的电子邮件系统如图 4-12 所示。

每个电子邮件都有一个邮箱地址，称为电子邮件地址（E-mail Address）。电子邮件地址可以是某个用户的通信地址，也可以是一组用户的地址。E-mail 地址的格式是固定的，并在全球范围内是唯一的。

用户的 E-mail 地址格式：用户名@主机名（如×××@mail. ×××.com）。主机名是指拥有独立 IP 地址的计算机域名，用户名是指在该计算机上为用户建立的 E-mail 账户名。

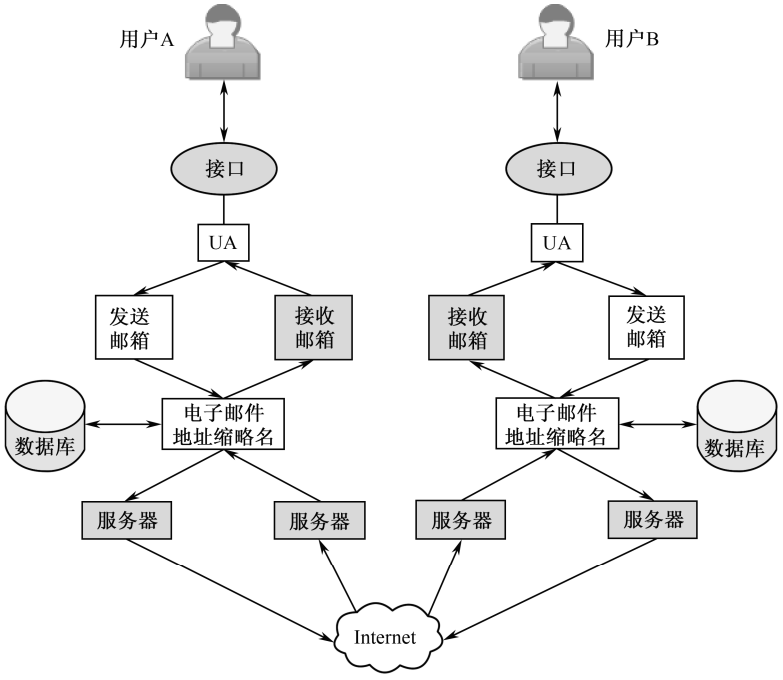


图 4-12　完整的电子邮件系统

4.5.3　电子邮件的格式

一个电子邮件分为信封和内容两大部分。Internet 文本报文格式只规定了邮件内容中的首部（Header）格式，而邮件的主体（Body）部分则由用户自己撰写。用户写好首部后，邮件系统会自动地将信封所需的信息提取出来并写在信封上，用户不需要填写电子邮件信封上的信息。E-mail 的组成如图 4-13 所示。

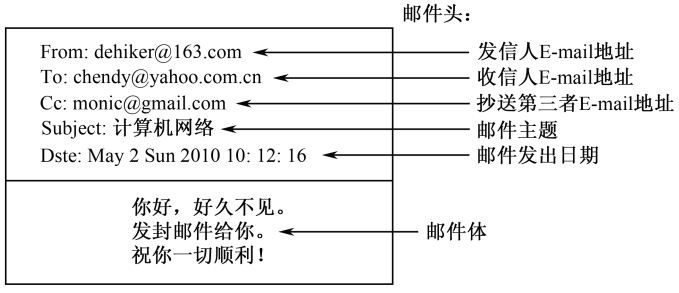


图 4-13　电子邮件的组成

邮件内容首部包括一些关键字，后面加上冒号。最重要的关键字是 To 和 Subject。

1）“To：”后面填入一个或多个收信人的电子邮件地址。在电子邮件软件中，用户将经常

通信的对象姓名和电子邮件地址写到地址簿（Address Book）中，当撰写邮件时，只需打开地址簿，单击收信人名字，收信人的电子邮件地址就会自动填入到合适的位置上。

2）“Subject：”是邮件主题。它是反映邮件的主要内容，便于用户查找邮件。

邮件首部还有一项是抄写“Cc：”，意思是留下一个副本，表示应给某人发送一个邮件副本。

3）邮件的首部还有“From：”和“Date：”，表示发信人的电子邮件地址和发信日期。这两项一般都由系统自动填入。

4.5.4　SMTP 简介

邮件服务器是电子邮件的核心组成部分，其功能是发送和接收邮件，并向发信人报告邮件的邮递情况，如成功发送、被拒绝、邮件丢失等。邮件服务器要实现其功能，需要使用两个不同的协议：一个是 SMTP 协议，用于发送邮件；另一个是邮局协议（Post Office Protocol，POP），用于接收邮件。

SMTP 是 TCP/IP 协议簇中的应用层协议，采用客户/服务器工作方式。当邮件服务器向另一个邮件服务器发送邮件时，该邮件服务器就作为 SMTP 客户；当邮件服务器从另一个邮件服务器接收邮件时，该邮件服务器就作为 SMTP 服务器。SMTP 协议通过 TCP 端口 25 提供服务。SMTP 协议的“邮件中继服务”可以跨网传输邮件，并实现不同类型的计算机之间电子邮件的发送。

一封电子邮件的发送和接收过程如下。

1）发信人调用用户代理，编辑要发送的邮件。

2）用户代理使用 SMTP 协议将邮件传送给发送端邮件服务器。

3）发送端邮件服务器将邮件存入邮件缓冲队列中，等待发送。

4）在发送端邮件服务器中，运行 SMTP 客户进程，当发现在邮件缓存中有待发送的邮件时，就向运行在接收端邮件服务器的 SMTP 服务器进程发起 TCP 连接建立请求。

5）当 TCP 连接建立之后，运行在发送端服务器中的 SMTP 客户进程就向远程的 SMTP 服务进程发送邮件。如果缓存中有多个邮件，则待所有的待发邮件一一发送完后，SMTP 关闭所建立的 TCP 连接。

6）在接收端邮件服务器中，运行的服务器接收到邮件后，将邮件放入收信人的用户邮箱中，等待收信人读取。

7）当收信人打算收信时，调用用户代理，使用 POP3（或 IMAP）协议，将自己的邮件从接收邮件服务器的用户邮箱中取出。

1．SMTP

SMTP 所规定的是两个相互通信的 SMTP 进程之间应如何交换信息，而不是问收信人和发信人是连接在同一个本地网络上的用户，还是 Internet 上其他网络的用户，或者是与 Internet 相连但不是 TCP/IP 网络上的用户。由于 SMTP 使用 C/S 模式，因此负责发送邮件的 SMTP 进程就是 SMTP 客户，而负责接收邮件的 SMTP 进程就是 SMTP 服务器。SMTP 的基本组成如图 4-14 所示。

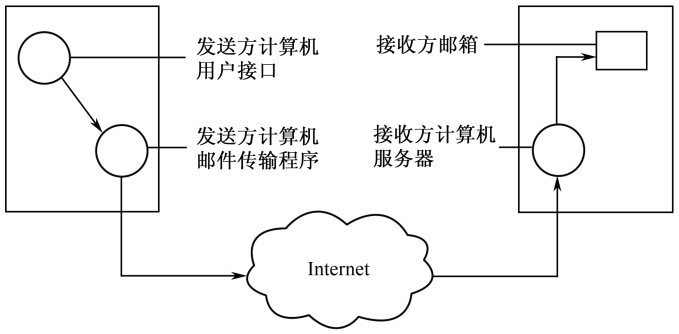


图 4-14　SMTP 的基本组成

（1）建立连接

SMTP 客户定时扫描邮件缓存，若发现有邮件，就使用 SMTP 的常用端口号 25 与目的主机的 SMTP 服务器建立 TCP 连接。这个 TCP 连接是发送端和接收端这两个邮件服务器之间直接建立的，不管发送端和接收端的邮件服务器相距多远，邮件传送过程中要经过多少个路由器。换言之，SMTP 不使用中间的邮件服务器。

在连接建立后，SMTP 服务器发出“220（域）service ready”，表示服务就绪。然后，SMTP 客户向 SMTP 服务器发送 HELO 命令，并附上发送方的主机名。若 SMTP 服务器有能力接收邮件，则回答“250”附上接收方的主机名，表示已准备好接收；若 SMTP 服务器不可用，则回答“421 service not available”。

（2）邮件传送

SMTP 客户发送 MAIL 命令，并附上发信人的地址。若 SMTP 服务器已准备好接收邮件，则回答“250 OK”；否则，返回一个代码，指出原因，如 452（存储空间不够）等。接着弄清楚接收端系统是否已做好接收邮件的准备。为此，SMTP 客户发送一个或多个 RCPT 命令，这取决于邮件的收信人是一个还是多个。其格式为“RCPT TO：<收信人地址>”。SMTP 客户每发送一个命令，SMTP 服务器都应返回相应信息。例如，“250 OK”表示收信人的邮件在接收端系统中，“550 No such user here”表示此邮箱不存在。

然后，SMTP 客户发送 DATA 命令，表示开始传送邮件的内容。SMTP 服务器返回“354 Start mail input；end with <CRLF>.<CRLF>”。这里<CRLF>的意思是“回车换行”。若不能接收邮件，则视情况返回应答代码，如 500（命令无法识别）等。

从此处开始为 SMTP 客户发送邮件的内容，内容发送完毕，再发送<CRLF>.<CRLF>，以表示邮件内容结束。若 SMTP 服务器正确收到了邮件，则返回信息“250 OK”；否则返回差错代码。

（3）邮件释放

邮件发送结束后，SMTP 客户发送 QUIT 命令，表示要释放连接。SMTP 服务器返回“221（域）service closing transmission channel”，同意释放 TCP 连接。到此邮件传送的全部过程结束。

上述 SMTP 客户与 SMTP 服务器的交互过程，对使用邮件的用户来说，被电子邮件系统的用户代理屏蔽了，是看不见的。

2．邮件读取协议 POP3 和 IMAP

现在常用的邮件读取协议有邮局协议第三版 POP3 和因特网报文存取协议（Internet Message Access Protocol，IMAP）两种。前者已成为 Internet 的标准，后者较新的版本是 IMAP4。这两种协议都是用户从目的服务器读取邮件所使用的协议，而且都使用 C/S 的工作方式。在接收邮件的用户 PC 中必须运行 POP3（或 IMAP）客户程序，而在其 ISP 的邮件服务器中则运行 POP3（或 IMAP）服务器程序，同时还必须运行 SMTP 服务器程序。

当用户拨号上网（或通过其他方式，如局域网等）连接成功后，就可以运行 POP3 客户程序。经 POP3 客户和 POP3 服务器之间交互一些命令与响应（对收信人都是透明的）后，POP3 客户程序便与 ISP 邮件服务器的 POP3 服务程序建立 TCP 连接。当用户输入鉴别信息（用户名和口令）后，POP3 服务器就允许对邮箱进行读取。

POP3 是一个脱机协议。因为从网上收到的邮件根据收信人的邮箱地址交付给目的 ISP 邮件服务器的 POP3 服务器后，收件人用 PC 不定期地连接到这个邮件服务器下载发送给他的邮件，并中断与 POP3 服务器的连接。一旦邮件交付给用户的 PC，POP3 服务器就不再保存这些邮件（事先设置除外），所有对邮件的处理都在用户的 PC 上进行。因此，POP3 服务器是一个具有存储转发功能的中间服务器。

而 IMAP 协议却不同，虽然 IMAP 也是将所有收到的邮件同样先送到 ISP 的邮件服务器的 IMAP 服务器，在用户的 PC 上运行 IMAP 客户程序，然后与 ISP 邮件服务器上的 IMAP 服务器程序建立 TCP 连接。但是，用户在自己的 PC 上，就可以像在本地操作一样操作 ISP 邮件服务器的邮箱。因此，IMAP 是一个联机协议。用户可以在 PC 上通过 IMAP 客户程序打开 IMAP 服务器的邮箱，查询和管理邮件。当用户需要打开某个邮件时，该邮件才传到用户的计算机上，而且 IMAP 还允许收信人只读取邮件中的某一部分。只要用户未发出删除邮件的命令，邮件就保存在 IMAP 服务器中。

3．通用 Internet 邮件扩充 MIME

由于电子邮件协议 SMTP 存在以下缺点，因此其需要进行扩充。

1）SMTP 不能传送可执行文件或其他二进制对象。

2）SMTP 只限于传送 7 位的 ASCII 码，许多非英语国家的文字（如中文、俄文）无法传送。

3）SMTP 服务器拒绝超过一定长度的邮件，但随着声音和图像的传送，邮件的长度在不断增大。

MIME（Multipurpose Internet Mail Extension）对 SMTP 的扩充，既不改动也不替代 SMTP，而是继续使用目前的格式，但增加了邮件主体的结构，并定义了传送非 ASCII 码的编码规则，所以 MIME 邮件可以在现有的电子邮件程序和协议下传送。MIME 的主要内容如下。增加了 5 个新的邮件首部字段，以提供有关邮件的主体信息。

1）MIME-Version：标识 MIME 的版本，现在的版本号是 1.0。若无此行，则为英文文本。

2）Content-Description：为可读字符串，说明此邮件是什么和类似邮件的主题。

3）Content-ID：邮件的唯一标识符。

4）Content-Transfer-Encoding：邮件的主题在传送时是如何编码的。

5）Content-Type：说明邮件的性质或类型。MIME 标准规定了 7 个基本的内容类型和 15 种子类型，使之能传送非英文文本、声音、图像、视频等多媒体信息。

4.6 万维网

4.6.1 万维网的工作原理

1．万维网简介

万维网又称环球信息网，英文简称为 WWW（World Wide Web）或 Web。

万维网并不是某一类型的计算机网络，实际上，万维网是 Internet 的一种应用系统，是一个大规模的提供海量信息存储和交互式超媒体信息服务的分布式信息系统。用链接的方法可以非常方便地从 Internet 上的一个 Web 站点访问另一个 Web 站点，从整个 Internet 上获取极为丰富的信息。图 4-15 说明了万维网提供分布式服务的特点。

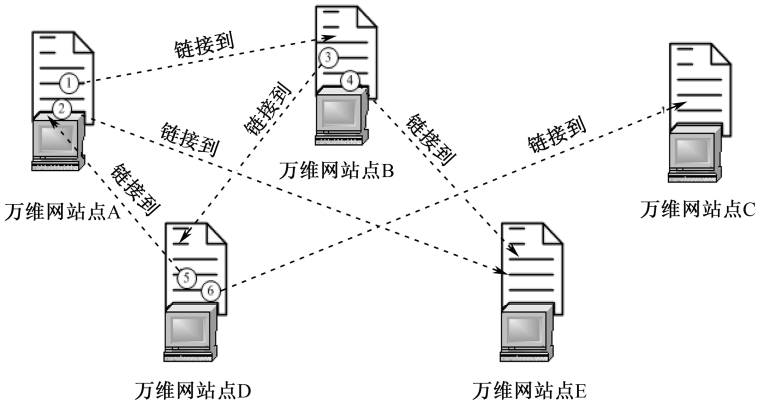


图 4-15　万维网提供分布式服务的特点

万维网是日内瓦的欧洲原子核研究委员会（法文缩写为 CERN）于 1989 年提出的。CERN 有几台高级加速器分布在几个国家的物理学家的实验室中，当初开发万维网的目的是使分布在这些国家的科学家们能更方便地交流信息，协同工作。但万维网成功的意义却远远不止于此，现在 Internet 上 Web 的应用远远超过其他应用，万维网是 Internet 发展中的一个重要里程碑。

万维网也是以 C/S 模式工作的。浏览器就是在用户计算机上的客户端程序，万维网文档所驻留的计算机运行服务器程序，即万维网服务器。万维网的这种基于 Wed 的 C/S 模式称为 B/S 模式。浏览器向万维网服务器发出信息浏览请求，服务器向客户送回客户所要的万维网文档，显示在客户的屏幕上，称为页面，其中默认的封面的万维网文档信息成为主页（Homepage）。

1993 年 2 月，第一个名为 Mosaic 的图形界面的浏览器开发成功，它的作者后来离开美国国家超级计算应用中心（NCSA）创办了 Netscape 通信公司。1995 年著名的 Navigator 浏览器面

市。现在使用最广泛的浏览器是 Netscape 公司的 Navigator 和 Microsoft 公司的 Internet Explorer。

万维网是一个分布式的超媒体（Hypermedia）系统，超媒体系统是超文本（Hypertext）系统信息多媒体化的扩充，超媒体是万维网的基础。Hypermedia 一词后缀“media”的意思是信息的载体，除了文本外，还可以是声音、图形、图像、动画及视频图像等表示方式。Hypermedia 这个词前缀“hyper”的意思是一个超媒体，是使用超链接（Hyperlink）将多个信息源链接而成的。超链接包含在每一个页面中，能够链接到其他万维网页面的链接信息。利用一个链接可以由一个文档找到一个新的文档，由这个新文档又可链接到其他文档，如此链接下去，可以在全世界范围内连接于 Internet 上的超文本系统中漫游。

为了标识分布在整个 Internet 上的万维网文档，万维网使用了统一资源定位符（Uniform Resource Locator，URL），使得每一个万维网文档在 Internet 的范围内都具有唯一的标识。

为了使万维网文档在 Internet 上传送，万维网使用超文本传送协议（Hyper Text Transfer Protocol，HTTP），客户和服务器程序之间的交互遵守 HTTP。HTTP 在 TCP/IP 体系中是一个应用层协议，基于传输层的 TCP 进行可靠的传输。

万维网文档使用超文本标记语言（Hyper Text Markup Language，HTML）编写，它支持使用超链接从本页面的某处方便地链接到 Internet 上任何一个万维网页面。

万维网文档可以划分为 3 种基本形式：静态文档（Static Document）、动态文档（Dynamic Document）和活动文档（Active Document），分别可用来显示静态的、动态的和活动的页面。

2．万维网的工作机制

（1）用浏览器访问 Web 服务器

万维网采用 C/S 模式，浏览器和服务器的交互式使用超文本传送协议（HTTP），它工作于 TCP 之上。浏览器访问 Web 服务器的工作机制：每个 Web 站点都持续不断地运行一个服务器进程，它通过 TCP 的周知端口 80 监听浏览器向它发出的连接请求。浏览器即客户进程。用户若要上网访问，浏览器就通过 URL 指向某个 Web 服务器来发出连接请求，该服务器监听到客户的连接请求后，建立起双方的 TCP 连接。然后，浏览器向服务器发送浏览某个页面的请求，服务器做出响应返回浏览器所请求的页面。最后，TCP 连接释放。图 4-16 给出了万维网工作机制的示意图（图中，HTTP 请求报文和响应报文的知识将在 4.5.3 节中介绍）。

如果浏览器的用户单击了网页上的一个可选择链接，它对应指向另外一个页面的一个超链接，假设该超链接的 URL 是 http://www.bistu.edu.cn/survey/bxinfo.htm，那么，处理过程如下。

1）浏览器分析页面的 URL。

2）浏览器向 DNS 请求解析服务器的域名 www.bistu.edu.cn 的 IP 地址，DNS 解析 IP 地址并做出应答。

3）浏览器使用服务器的 IP 地址和周知端口号 80 与服务器建立 TCP 连接。

4）浏览器发出取文件 HTTP 命令：GET/servery/bxinfo.htm。

5）服务器响应，将文件 bxinfo.htm 发送给浏览器。

6）双方释放 TCP 连接。

7）浏览器显示文件 bxinfo.htm 的页面。

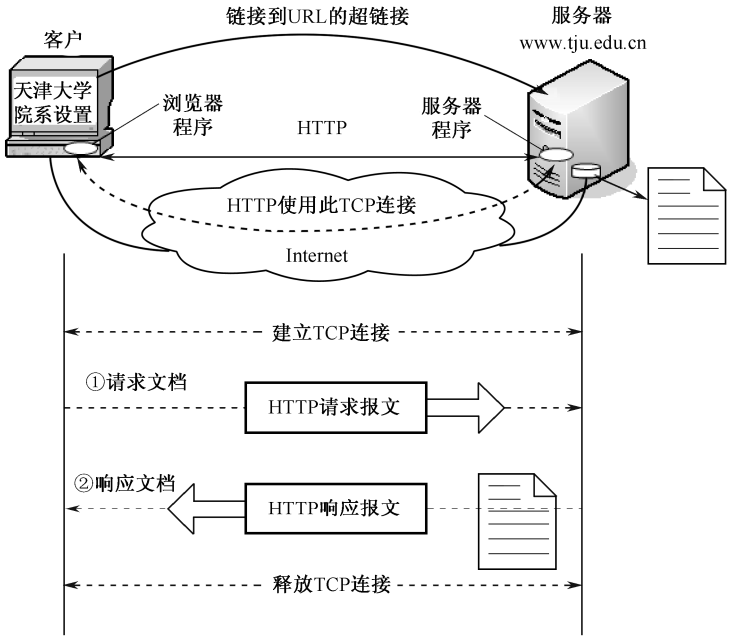


图 4-16　万维网的工作机制

（2）万维网代理

可以使用万维网代理（Web Proxy）进行万维网的访问。Web 代理一般是运行在本地 LAN 上的一台主机上的一个进程，它代理用户万维网的访问，许多 ISP 也运行万维网代理。运行万维网代理主机的磁盘上存储了大量的近期访问 Internet Web 服务器所得到的网页的复制内容，可以在后来的同样访问中使用它们，这种技术称为缓存（Cashing），这是在 ARP 和 DNS 中也使用的技术。

为了使用 Web 代理技术，浏览器也要做出相应的配置，使得所有的页面访问请求都发送给代理。使用 Web 代理的工作过程如下：浏览器访问万维网时，先与代理建立 TCP 连接，并向它发出 HTTP 请求报文。如果运行代理的主机磁盘中已经存储了该请求的对象，则将此对象放到 HTTP 响应报文中返回给浏览器；否则，代理就代表该浏览器与 Internet 上的源头服务器（Origin Server）建立 TCP 连接，并发出 HTTP 请求报文。代理从源头服务器收到这个请求的对象后，先复制到自己的磁盘中，以便今后使用，然后放在 HTTP 响应报文中发送给原请求该对象的浏览器。可见，Web 代理既作为客户又作为服务器。

（3）浏览器访问 FTP 服务器

除 Web 服务器外，浏览器还可以访问其他类型的服务器（如 FTP 等）。当浏览器访问其他类型的服务器时，工作过程有所不同。这里有两种方式，以 FTP 为例分别绘于图 4-17（a）和图 4-17（b）中。

图 4-17（a）所示的浏览器除了含有 HTTP 客户外，还含有 FTP 客户，它直接使用 FTP 客户访问 FTP 服务器，访问过程使用的是 FTP 协议。

图 4-17（b）使用一个代理服务器软件，它是一种网关，浏览器只含有 HTTP 客户，它用

HTTP 协议与代理服务器交互，而代理服务器将浏览器的请求翻译成 FTP 请求和 FTP 服务器交互，使用 FTP 传送协议。对于 FTP 服务器的应答，代理服务器同样进行一次逆向的转换。代理服务器可以和浏览器运行在同一台机器上，也可以运作在网络上的一台独立的机器上为多个浏览器服务。代理服务器还可以进行缓存，保存经过它的网页以减少网络通信量。

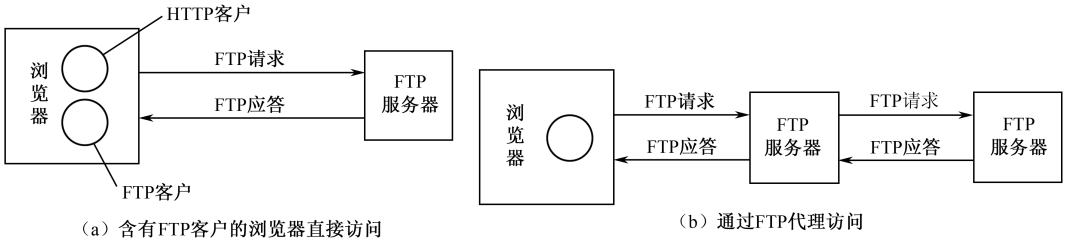


图 4-17　浏览访问 FTP 服务器

（4）搜索引擎

万维网是一个信息的大千世界，有着海量的信息资源，分布在全球数以百万计的 Web 服务器上。因此，如何能在如此浩瀚无际的信息海洋里，快捷、方便地查找到自己所需要的信息，是一个非常困难而重要的问题。搜索引擎（Search Engine）就是为此而产生的信息搜索工具。搜索引擎以万维网页面标题或内容中的关键词作为索引，将搜索到的相关链接返回给用户。著名的搜索引擎有 Google（www.google.com）、Yahoo（www.yahoo.com）和中文搜索引擎百度（www.baidu.com）等。

3．浏览器

万维网浏览器的结构比较复杂，图 4-18 是浏览器的结构框图。图中粗的空心箭头表示数据流向，细箭头表示控制关系。图中虽然未画出客户和相应解释程序之间的数据联系，但这种联系是存在的。

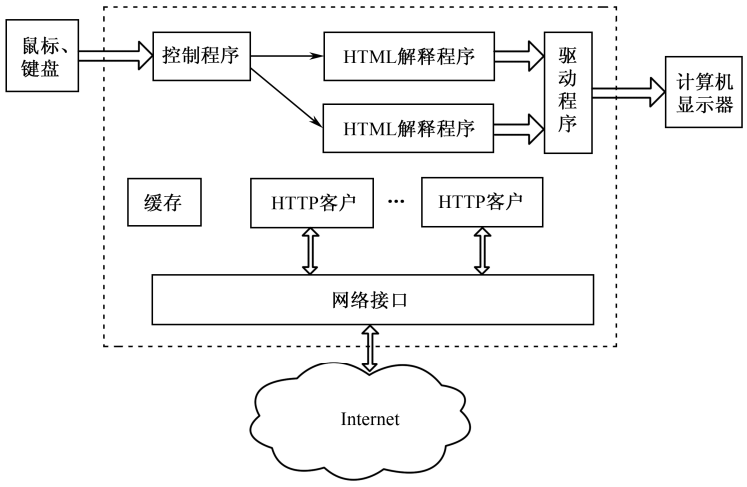


图 4-18　浏览器的结构框图

一个浏览器主要包括一组客户、一组解释程序及一个控制程序。控制程序是核心部件，它管理调度客户和解释程序，解释单击和键盘的输入，调用有关的程序来执行相应的操作。例如，当用户单击一个超链接的起点时，控制程序就调用一个客户从远处的服务器上取回该文件，并调用相应解释程序进行解释，最终由显示驱动程序驱动，显示该文档的页面。

浏览器必须包含 HTTP 客户，HTTP 客户用来与服务器建立连接和交换数据。浏览器还可以包含一个 FTP 可选客户，用来获取文件传送服务。一些浏览器还包含一个电子邮件可选客户，使浏览器也能够发送和接收电子邮件。浏览器屏蔽了许多细节，用户并不能感觉到它执行了一个可选客户。

HTML 的解释程序是必需的，而其他的解释程序则是可选的。解释程序对 HTTP 客户从服务器得到的 HTML 文档进行解释，并将其转换为适合用户显示硬件的命令来处理版面的细节，显示驱动程序将页面在显示器上显示出来。

如图 4-19 所示，在浏览器中还可设一个缓存，浏览器将它取回的页面副本都存入本地磁盘的缓存中。当用户浏览某个页面时，浏览器首先检查本地的缓存。若缓存中保存了该页面，浏览器就直接从缓存中读取该页面，而不必通过网络得到，因而明显地改善了浏览器的运行速度。但问题的另一面是，如果缓存中保存的是用户今后不再感兴趣、不再浏览的页面，缓存不但不会改善性能，反而会因为徒劳地进行磁盘操作而降低了浏览器的性能。因此浏览器一般允许用户调整缓存策略。例如，用户可以设置页面缓存时间的时限等。

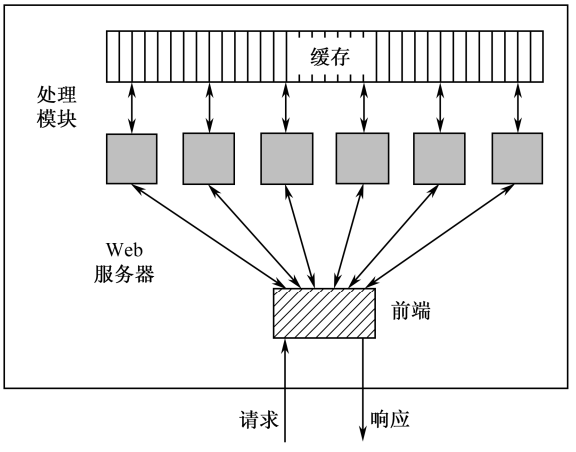


图 4-19　使用缓存的多线程 Web 服务器

4．Web 服务器

在万维网中，服务器执行的任务相对比较单纯：等待浏览器请求，建立 TCP 连接，根据浏览器发来的请求从磁盘读取文件并发回浏览器，然后关闭连接，再等待下一个请求。

但这里存在一个问题：服务器的响应速度受到磁盘访问时间的限制。高端的 SCSI 接口磁盘的平均访问时间是 5ms 左右，这就限制了 Web 服务器每秒最多可处理 200 次请求。如果常常要读取大的文件，那么处理的请求次数就更少。对于一个大的 Web 站点，这样的速度不能满足要求。

一种常用的改进方法是在内存中维护一个缓存，其中保存着最近访问过的文件，服务器在磁盘读取文件之间先检查缓存，如果缓存有该文件，就从缓存中直接读取，避免了磁盘访问。

进一步的措施是使服务器变为多线程模式。图 4-19 所示了使用缓存的多线程 Web 服务器的一种方案。服务器由一个前端模块和 n 个处理模块组成，共 n+1 个线程同属于一个进程，所有的处理模块都可以访问当前进程地址空间中的缓存。前端模块接收所有进来的浏览请求，将请求交给某一个处理模块进行处理。处理模块首先检查缓存，如果缓存中存有该文件，则

从缓存中直接读取并通过前端模块将它发给用户。如果缓存中没有该文件，则处理模块读取磁盘，将文件存入缓存并通过前端模块将它发回给用户。当一个或多个处理模块因为等待磁盘操作而被阻塞时，其他处理模块可以继续处理其他请求。在多线程的基础上，还可以使用多磁盘，它们能同时工作，进一步提高了处理速度。

图 4-20（a）所示的 Web 服务器场（Server Farm）方案使用更多的计算机进行 Web 服务处理。前端节点仍然接收所有进程的浏览请求，但它将请求分散给多个计算机（CPU）进行处理而不是多个线程，每台计算机可以使用多线程模式。前端节点还可以记录每个请求转发到哪个处理节点，并将后续的同样请求都转发到该节点，这样使得某个节点专门处理某些页面，从而不会因每个节点的缓存中都保存同样的页面文件而浪费缓存空间。

服务器场的一个问题是客户的 TCP 连接请求都进入前端节点，响应也都经过前端节点，影响处理速度。一种称为 TCP 移交（TCP Bandoff）的技术可以解决这个问题。通过 TCP 移交，TCP 端点可以被传递到处理节点上，处理节点可以直接给客户发回响应，不再经过前端节点，如图 4-20（b）所示。

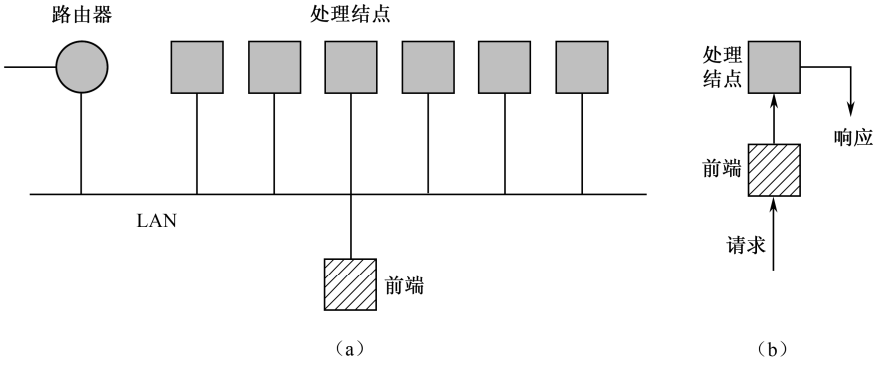


图 4-20　Web 服务器场与 TCP 移交

4.6.2　统一资源定位符（URL）

1．URL 的格式

统一资源定位符（Uniform Resource Locator，URL）[RFC1738、1808，建议标准]给 Internet 上的资源的位置和访问方式提供一种抽象的表示方法。对于与 Internet 相连的机器上的任何可访问的对象来说，URL 是唯一的，可以将 URL 想象为一台计算机上的文件名系统在整个 Internet 范围的扩展。

统一资源定位符中的“资源”是指在 Internet 上可以被访问的任何对象，包括文件目录、文件、文档、图像、声音等及与 Internet 相连的任何形式的数据。“资源”还包括电子邮件的地址。

URL 不仅可以用于漫游万维网，而且也可以用于 FTP、E-mail 和 Telnet 等，这样将几乎所有 Internet 访问统一为一个程序，即万维网浏览器。

URL 的格式如下：“访问方式：//服务器域名[：端口号]/路径/文件名”。

URL 一般使用小写字母，但它对字母大小写不敏感。URL 的前面（冒号左边）部分指明了 URL 的访问方式。URL 可使用多种访问方式，如：

1）http：超文本传送协议（HTTP）。

2）ftp：文件传送协议（FTP）。

3）telnet：用于交互会话。

4）mailto：电子邮件地址。

2．使用 HTTP 的 URL

对于万维网网点的访问要使用 HTTP 协议。HTTP 的 URL 的一般形式为“http：//服务器域名[：端口号]/路径/文件名”。

HTTP 的默认端口号是 80，可以省略。如果 URL 在服务器域名后使用了非默认的端口号，就不可以省略。HTTP URL 中的“路径/文件名”用于直接指向服务器中的某一个文件。例如，http://www.bistu.eud.cn/servey/beinfo.htm 是北京信息科技大学 www 主机中目录/servey/下的一个关于北京信息科技大学简介的文件 bxinfo.htm，文件后缀“.htm”表示该文件是用 HTML 语言编写的。如果省略路径和文件名，则 URL 就指向了 Internet 上的某个主页。例如，使用如下的 URL 即可进入北京信息科技大学的主页：“http://www.bistu.edu.cn”。

3．使用 FTP 的 URL

使用 FTP 访问站点 URL 的最简单的形式是使用主机域名（也可以使用 IP 地址）。例如，“ftp://rtfm.mit.edu”。其中，rtfm.mit.edu 是麻省理工学院 MIT 的匿名服务器 rtfm 的 Internet 域名。如果我们要直接访问上述服务器中存于 pub 下的一个文件 readme.txt，那么该文件的 URL 就是“ftp://rtfm.mit.edu/pub/readme.txt”。

除 URL 之外，还有另一个通用的万维网标识符，即通用资源标识符（Universal Resource Identifier，URI）。请注意，这里的 U 是 Universal 而不是 Uniform 的缩写。URI 的规格说明[RFC1630]定义了对任意命令和编址方式进行编码的语法。URI 的概念和诸多细节还在发展之中。

URL 机制有一个固有的弱点，就是 URI 必须指向一个特定的主机。如果有的页面被频繁地进行访问，那么自然就希望将此页面复制几份存放在不同地点，这样就可以减少网络的通信量。但是有 URL 时必须指明该页面的所在地。例如，我们不能说“我希望访问页面 abc，但我不想知道此页面在何处”。要解决这个问题就要用到 URI，因为 URI 的好处是它使一个资源的名字与其位置无关，甚至与访问的方法无关。URI 包括了 URL 和统一资源名字（Uniform Resource Name，URN）。因此 URI 可看成是一种广义的 URL。而 URL 只是 URI 的一种类型，在 URL 中指明了访问的协议及一个特定的 Internet 地址。

4.6.3　超文本传送协议（HTTP）

1．HTTP 简述

超文本传送协议（HTTP）是万维网上能够可靠地交换各种多媒体信息的基础，是 Web 的核心。目前使用的版本多为 HTTP 1.0[RFC 1945，草案标准]和 HTTP 1.1[RFC 2616，草案标准]。

HTTP 是 TCP/IP 体系中应用层的协议，它基于传输层的 TCP。HTTP 服务器通过 TCP 的周知端口 80 监听客户向它发出的连接请求。TCP 可以实现面向连接的可靠的传输，而 HTTP 本身并没有重传等可靠性机制。

HTTP 1.0 为每次请求都建立一次 TCP 连接，服务器发回响应后 TCP 连接就被释放，这是一种非持续连接。

HTTP 1.0 协议是无状态（Stateless）的，也就是说，同一个客户第二次访问同一个服务器上的相同页面时，服务器的响应与第一次被访问时的相同，因为服务器不记得曾经访问过这个客户，也不记得曾经服务过多少次。

HTTP 1.1 支持持续连接（Persistent Connection）并把它作为默认选择，对于用户连续的多个访问请求，TCP 连续不被释放，这可减少开销，提高效率。

2．HTTP 的报文结构

HTTP 有两类报文：HTTP 客户的请求报文；服务器的响应报文。

HTTP 规定 HTTP 客户和服务器交互的是 ASCII 码文本的请求和类 MIME（MIME-like）的响应。

图 4-21 是 HTTP 两种报文的结构，图中阴影部分表示空格，CRLF 表示回车换行。

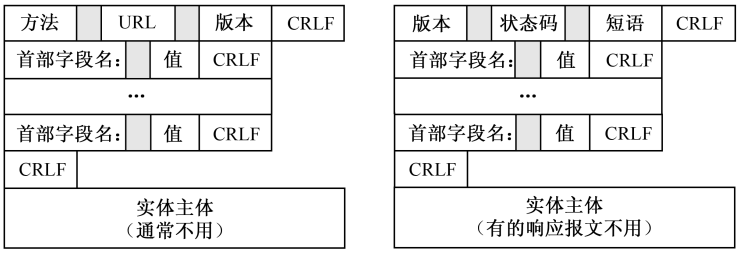


图 4-21　HTTP 的报文结构

如图 4-21 所示，请求报文和响应报文都是由 3 部分组成的。第一行分别为请求行和状态行，最后一行都为实体主体，中间的都为首部行。可以看出，这两种报文格式的区别就是开始行不同。

下面简单介绍这 3 个部分。

1）请求行和状态列。请求报文中的第一行是请求行，它有 3 个内容：方法（Method）、请求资源的 URL 及 HTTP 的版本。所谓方法，就是对所请求的对象进行的操作。下面是几种常见的方法。

- GET：最常使用的命令，请求读取 URL 所标识的页面。
- HEAD：与 GET 相似，但请求读取的只是页面的首部。
- PUT：与 GET 的功能相反，PUT 操作是存入一个页面，可用于新增和更改页面。使用 PUT 命令要检验请求首部行中的授权（Authorization），未得到授权的人不得随便使用 PUT 操作。
- POST：与 PUT 类似，但它是把数据附加到原来页面的后面。
- DELETE：请求服务器删除在请求行中 URL 所标识的资源。使用 DELETE 命令也需

要授权。

- COPY：将请求行中 URL 所标识的资源复制到另一个网点。

响应报文段的第一行是状态行，包含 3 项内容：HTTP 的版本、状态码（Status-Code）及解释状态码的短语。状态码由 3 位数字组成，分为以下 5 类。

- 1××：表示通知信息，如 100 表示服务器同意客户的请求。
- 2××：表示成功，如 200 OK 表示请求成功，204 表示没有内容存在。
- 3××：表示重定向，如 301 Moved Permanently 表示请求对象已永久转移，新的 URL 在本响应报文的 Location 首部指出。
- 4××：表示客户的差错，如 400 Bad Request 表示服务器无法理解客户的请求，404 表示页面未找到。
- 5××：表示服务器的差错，如 505 HTTP Version Not Supported 表示服务器不能支持请求的 HTTP 版本。

2）首部行。首部行用来说明浏览器、服务器和报文主体的一些信息。首部行的行数在不同情况下可以不同。在第一首部行中有首部字段名和它的值，每一行在结束的地方都要有“回车”和“换行”。整个首部行结束后还要有一空行和后面的实体主体分开。常用的首部字段名如下。

- User-Agent：用于请求报文，客户将其浏览器、操作系统等属性信息告知服务器。
- Accept：用于请求报文，指出什么 MIME 类型是可以接受的。
- Accept-Charset：用于请求报文，指出什么字符集是可以接受的。
- Accept-Encoding：用于请求报文，指出什么编码方式是可以接受的。
- Accept-Language：用于请求报文，指出浏览器希望接收的自然语言。
- Keep-Alive：用于请求报文，指出一个最长时间或最大请求数目，其间可保持 TCP 持续连接。
- Date：用于请求/响应报名，指出报文发送的日期和时间。
- Server：用于响应报文，指出关于服务器的信息。
- Content-Encoding：用于响应报文，指明实体主体的编码方式。
- Content-Length：用于响应报文，指明以字节为单位的实体主体的长度。
- Content-Type：用于响应报文，指明实体主体采用的 MIME 类型。
- Location：用于报文响应，告诉客户应该尝试另一个 URL。

3）实体主体。请求报文一般不包含实体主体，响应报文的实体主体可包含任意长度的字节序列，HTTP 能够传送多媒体类型的内容，类 MIME 实体在浏览器应如何解释，取决于相关首部行的说明。

3．HTTP 请求报文实例

下面是一个请求报文的例子。

```
GET/chn/yxsz/index.htm    HTTP/1.1   //请求行使用了相对 URL
Host:   www.bistu.edu.cn              //此行是首部行的开始，给出主机的域名
Connection:   close                   //告诉服务器发送完请求的文档后就可释放连接
User-Agent:    Mozilla/5.0            //表明用户代理使用 Netscape 浏览器
```

```
Accept-Language：  cn                    //表示用户希望优先得到中文版本的文档
                                         //请求报文的最后还有一个空格
```

在请求行使用了相对 URL（即省略了主机的域名）是因为下面的首部行（第 2 行）给出了主机的域名。第 3 行告诉服务器不使用持续连接（Persistent Connection），表示浏览器希望服务器在传送完所请求的对象后即关闭 TCP 连接。这个请求报文没有实体主体。

4.6.4　超文本标记语言（HTML）

1．HTML 简介

现在计算机使用的字处理器种类繁多且版本各异，在某一台计算机屏幕上显示出的文件，在另一台计算机上未必能显示处理。万维网要使任何一台计算机都能显示出任何一个万维网服务器上的页面，就必须解决页面制作的标准化问题。超文本标记语言就是一种制作万维网页面的标准语言，它消除了不同计算机之间信息交流的障碍。

超文本标记语言（Hyper Text Markup Language，HTML）中 Markup 的意思就是“设置标记”，这些标记指明信息显示的格式，如在何处用何种字体显示等，因此也可以将 HTML 译为超文本排版语言。

ISO 早在 1986 年就已制定了一个标准 ISO 8879，即 SGML（Standard Generalized Markup Language）。这是一个描述标记语言的标准[W-SGML]。SGML 是一个非常复杂的、功能丰富的系统，有很多种选项，很适合于需要精确文档标准的大型组织。然而 SGML 的过分复杂使它很不适合于简单快捷的万维网使用。由于 HTML 非常易于掌握且实施简单，因此它很快就成为万维网的重要基础[RFC 1866]。官方的 HTML 标准由 W3C（即 WWW Consortium）负责制定。

HTML 定义了许多用于排版的命令，即“标签（Tag）”。例如，<I>表示后面开始用斜体字排版，而</I>则表示斜体字排版到此结束。HTML 将各种标签嵌入到万维网的页面中，就构成了所谓的HTML 文档。HTML 文档是一种可以用任何文本编辑器（如Windows 的记事本Notepad）创建的 ASCII 码文件。但应注意，仅当 HTML 文档以.html 或.htm 为扩展名时，浏览器才对这样的 HTML 文档的各种标签进行解释。如果 HTML 文档以.txt 为其后缀，则 HTML 解释程序就不对标签进行解释，而浏览器只能看见原来的文本文件。

当浏览器从服务器读取某个页面的 HTML 文档后，就按照 HTML 文档中的各种标签，根据浏览器所使用的显示器的尺寸和分辨率大小，重新进行排版，并恢复出所读取的页面。

目前已开发出了很好的制作万维网的软件工具，使我们能够像使用 Word 字处理器那样很方便地制作各种页面。然而学习一些 HTML 的基本概念仍是必要的，这是因为在对已有的万维网页面进行修改时，往往要查看其源代码，即查看其 HTML 文档。直接在 HTML 文档上对页面进行修改，有时是很必要的。

2．HTML 的格式与标签

HTML 文档由两个主要部分组成：首部（Head）和主体（Body）。首部在文档的前面，包含文档的标题（Title）等。当浏览器显示 HTML 页面时，文档的标题显示在最上面的标题条中。

文档的主要信息包含在主体中。HTML 文档主体部分由若干个更小的元素（Element）组成，如段落（Paragraph）、表格（Table）和图像（Image）等。

HTML 定义了许多标签（Tag），用于说明排版的格式。HTML 用一对标签或几对标签来标识一个元素。一对标签包括一个开始标签和一个结束标签。开始标签由一对尖括号和标签名组成。表 4-3 的每一行即是一对标签，其中尖括号内是标签名，不区分大写和小写。结束标签在标签名前面加了一个“/”。有一些标签可以省略结束标签。

表 4-3　常用的 HTML 标签

标　签	描　述
<HTML>...</HTML>	声明是用 HTML 编写的万维网文档
<HEAD>...</HEAD>	页面首部
<TITLE>...</TITLE>	定义标题，不在浏览器的显示窗口显示
<BODY>...</BODY>	页面主体
<Hn>...</Hn>	*n* 级标题，*n*=1～6，1 级最高
<B>...</B>	设置为粗斜体
<I>...</I>	设置为斜体
<UL>...</UL>	设置为无序的列表
<OL>...</OL>	设置为有序的列表
<MENU>...</MENU>	设置为菜单
<LI>	表项的开始（不可用</LI>）
 	换行
<P>	一段开始
<HR>	水平线
<PRE>...</PRE>	预格式化文本，浏览器显示时不需要重新排版
<IMG SRC= ″ ... ″ >	装载图像文件
<A HREF= ″ ... ″ >...</A>	定义超链接

下面是一个简单的例子，用来说明 HTML 文档中标签的用法。

```
<HTML>                                  //HTML 文档的开始
<HEAD>                                  //首部开始
    <TITLE>一个 HTML 的例子</TITLE>      //“一个 HTML 的例子”是标题
</HEAD>                                  //首部结束
<BODY>                                  //主体开始
    <H1>HTML 很容易掌握</H1>             // “HTML 很容易掌握”是一级标题
    <P>这是第一段落。   虽然很           //<P>和</P>之间的文字是一个段落
    短，但它仍是一个段落。</P>           //实际上，标签</P>经常可省略不用
    <P>这是第二个段落。</P>              // <P>和</P>之间的文字是一个段落
</BODY>                                  //主体结束
</HTML>                                  //HTML 文档结束
```

HTML 可以在页面中插入插入表格，插入表格要使用标签<TABLE>。与<TABLE>标签配套使用的还有其他标签，用来说明表格的细节，如<CAPTION>（表格的标题）、<TR>（表格的行）等。

HTML 支持在页面中插入图像，标题<IMG>表明在当前位置插入一幅图像。HTML 没有规

定图像的格式，但大多数的浏览器都支持 GIF（Graphic Interchange Format）文件和 JPEG（Joint Photographic Experts Group）文件，它们都是位图（Bit Map）文件（.bmp），使用不同亮度、色调的像素矩阵来创建图。在插入图像时，在标签<IMG>中可以使用一些参数进行具体的说明。例如，SRC 参数给出了图像文件的 URL，参数 ALIGN 给出了图像定位，参数 HEIGHT 和 WIDTH 指明图像装入时在屏幕上显示尺寸的大小，一般用像素（Pixel）的数目表示。例如，<IMG SRC= portrait.gif HEIGTH=100 WIDTH=65>表示装入一个文件名为 portrait.gif 的图像，其高度和宽度分别为 100 和 65 像素。

3．HTML 的超链接

超链接对于万维网至关重要，没有超链接就没有万维网。超链接标签是最重要的一个 HTML 标签。

（1）远程链接和本地链接

HTML 定义超链接的标签<A HREF=″…″>…</A>，字符 A 表示锚（Anchor），比喻建立一个超链接好像抛出一个锚，这个锚扎到超链接的终点。每个超链接都有一个起点和终点。超链接的起点表示一个超链接在万维网页面中从何处引出，它可以是一个页面中的一个字符串（含一个字符）或一幅图像等，单击它们，就从该处出发链接到一个新的页面。终点用这个新页面的 URL 表示。

在 HTML 文档中定义一个超链接的语法如下。

```
<A HREF=”Terminal-URL”>start</A>
```

式中，start 是超链接的起点，如果起点是字符串，start 就是该字符串；如果起点是一幅图，start 还要使用图像文件的标签<IMG SRC=″…″>，图的文件名放在引号中。Terminal-URL 是超链接终点的统一资源定位符，放在 HREF=″…″的引号中。HREF 与 A 中间有一个空格。HREF 中，“H”代表超文本，“REF”代表 Reference，是“引用”的意思。

如果超链接的终点不在本网站上，而是其他网站上的页面，这种链接方式称为远程链接。如果超链接指向本计算机的某一个文件，这种链接方式则属于本地链接。

进行本地链接时，终点 Terminal-URL 不需要写完整的 URL。使用 URL 时，根据具体情况可以进行不同程度的简化。当协议（http://）、服务器域名、目录路径和文件名分别被省略时，当前页面的协议、当前的服务器、当前的目录路径和当前文件分别为默认值。

对于一个远程链接，如果 URL 中不包含路径和文件名，一般指服务器上的默认文件 index.html，通常为一个主页。

上述简化的方法中，终点 Terminal-URL 使用的是相对路径名。通过完整的 URL 寻找一个文件时可以使用绝对路径名。

（2）链接到本文件的某个地方

命名锚（Named Anchor）是 HTML 链接到同一个文件中某个位置的一种链接方法。在一个很长的万维网文档中，当需要查找其中的某些内容时，往往要利用滚动条在成百上千行的信息中反复查找，操作很不方便。对于这种情况，一个比较好的解决方案是：在文件的开始设计一个索引目录，目录中的每一项都是一个超链接的起点。超链接的终点则是同一个文件中被指明

的特定位置。HTML 将这种链接的终点称为命名锚，每一个链接终点有一个不同的命名，以区分多个链接终点。下式用来定义一个命名锚。

```
<A NAME=”Name Anchor”>terminal-characters</A>
```

其中，NAME 后面引号中的 Name Anchor 写入的是命名锚的名称。结束标签</A>前面的 terminal-characters 具体指明该链接的终点位置，terminal-characters 是这个位置开始的一个字符串（含一个字符）。

链接到一个命名锚的语法如下。

```
<AHREF=”#Name Anchor”>start</A>
```

其中，字符“#”后面的 Name Anchor 就是命名锚的名称。

使用命名锚也可链接到本地的其他 HTML 文件上，这时式中的字符“#”前应加上该文件名称，但命名锚不能链接到其他网点的文件上。

以上我们简要介绍了 HTML，它也存在一些问题。HTML 把内容和结构信息混合在一起，因此某些情况下应用不方便。有两种新的语言被开发出来，对 HTML 进行了改进和增强，使得 Web 页面可以被结构化，便于自动处理，一是可扩展标记语言（eXtensible Markup Language，XML），用一种结构化的方法来描述 Web 的内容；二是可扩展样式语言（eXtensible Style Language，XSL），以一种独立于内容的方式来描述格式。

4.6.5 动态 Web 文档技术

万维网文档可以划分以下 3 种基本形式。

1）静态文档（Static Document）。静态文档是最基本的万维网文档。静态文档创作完毕后存放在万维网服务器中，在用户浏览时，页面内容不会改变，只有程序员修改了静态文档，显示页面才会改变。

2）动态文档（Dynamic Document）。与静态文档不同，动态文档所看到的页面内容可以反映当时的变化，如股市行情等。动态文档的屏幕刷新由服务器完成。

3）活动文档（Active Document）。活动文档比动态文档有更快的刷新能力，可以连续、快速地进行显示屏幕的刷新，如动画等。活动文档的屏幕刷新由浏览器在本地实现。

1．动态文档的概念

动态文档的内容是在浏览器访问万维网服务器时才由应用程序动态创建的。当浏览器请求到达时，万维网服务器要运行另一个应用程序，并将控制转移到此应用程序。接着，该应用程序对浏览器发出的数据进行处理，并输出 HTTP 格式的文档，万维网服务器将应用程序的输出作为对浏览器的响应。由于对浏览器每次请求的响应都是临时生成的，因此用户通过动态文档所看到的内容可根据需要不断变化。可见动态文档的主要优点是具有报告当前最新信息的能力。例如，动态文档可用来报告股市行情、天气预报或民航售票情况等内容。但动态文档的创建难度比静态文档高，因为动态文档的开发不是直接编写文档本身，而是编写用于生成文档的应用程序。

动态文档和静态文档之间的主要差别体现在服务器部分，这主要是因为文档内容的生成方法不同。而从浏览器的角度来看，这两种文档并没有区别。动态文档和静态文档的内容都遵循 HTML 所规定的格式，浏览器仅根据在屏幕上看到的内容并无法判定服务器送来的是哪一种文档，只有文档的开发者才知道。

从以上所述可以看出，要实现动态文档就必须在以下两个方面对万维网服务器的功能进行扩展。

1）应增加另一个应用程序，用来处理浏览器发来的数据，并创建动态文档。

2）应增加一个机制，使万维网服务器将浏览器发来的数据传送给这个应用程序，然后万维网服务器能够解释这个应用程序的输出，并向浏览器返回 HTML 文档。

2．CGI 技术

通用网关接口（Common Gateway Interface，CGI）是实现动态文档的一种典型技术。为实现动态文档，CGI 从以下两个方面对万维网服务器进行了改进。

一方面，增加了一个应用程序，称为 CGI 程序，用来处理浏览器发来的数据并创建动态文档。浏览器访问万维网服务器时可以启动 CGI 程序，CGI 程序对浏览器发来的数据进行处理，并即时生成 HTML 格式的文档，万维网服务器将此文档作为对浏览器的响应发回浏览器。由于对浏览器每次请求的响应都是即时生成的，因此通过动态文档所看到的页面内容是变化中的当前最新信息。例如，动态文档可用来报告股市行情或民航铁路售票情况等经常变化的内容。

另一方面，增加了一个机制，万维网服务器和 CGI 程序通过它进行交互。增加的这个机制就称为 CGI。CGI 是一种标准，它规定了万维网服务器如何与这个新增的 CGI 程序交互。图 4-22 是扩充了功能的万维网服务器示意图。

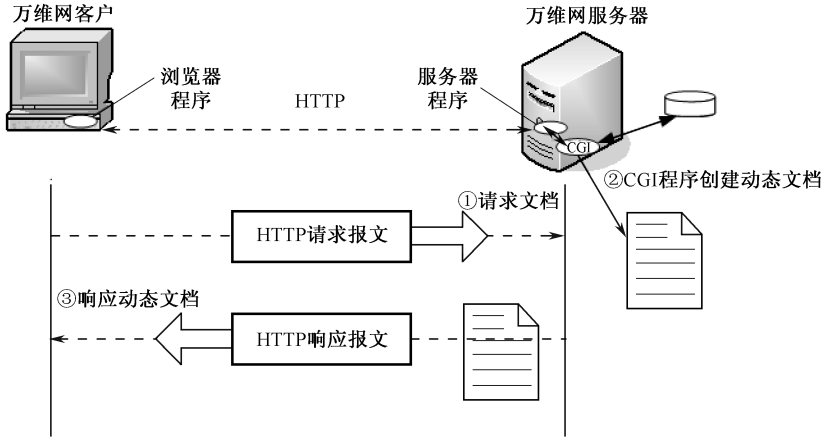


图 4-22　扩充了功能的万维网服务器

新增加的 CGI 程序的正式名称称为 CGI 脚本（CGI Script）。脚本一般指的是一个程序，它的特点是需要由另一个解释程序解释执行。就像电影和戏剧脚本，往往需要通过导演的解释（说戏）再进行表演。因为脚本需要解释执行，运行起来较一般的编译程序慢，但脚本可

以更容易地编码，适合于一些功能有限的小程序。CGI 并没有指定特定的编程语言。有一些语言专门作为脚本语言，如 Perl、JavaScript 等，常用的 C、C++等也可以编写脚本。CGI 脚本又称为 cgi-bin 脚本，这是因为在早期的程序中，所有的 CGI 脚本都放在目录/cgi-bin 下，它就成为 CGI 脚本 URL 的路径。

CGI 脚本驻留在万维网服务器上，当 CGI 脚本被调用时，服务器将一些参数传递给它，参数的值一般由浏览器提供。这样，在不同的情况下使用不同参数，就可以用一个 CGI 脚本产生细节不同的动态文档。

CGI 最初的设计是运行在 UNIX 操作系统的平台上，万维网服务器将参数传递给 CGI 脚本的方法，是将这些参数置于 UNIX 的环境变量中，CGI 脚本再从环境变量中将参数值读取出来，这样就实现了参数的传递。CGI 定义的环境变量有 18 个，它们被称为元变量（Metavariable），意思是独立于操作系统。具体实现时，不同的操作系统可以根据自己的情况采用不同的方法向 CGI 脚本传递参数。

CGI 脚本由来自浏览器的请求激活。例如，Web 页面中某个超链接中的 URL 指向一个 CGI 应用。

```
<A HREF=”http: //www.website.com/cgi-bin/cgiprog”>
```

其中，cgiprog 是万维网服务器 www.website.com 上的一个 CGI 脚本，放在目录/cgi-bin 下。当浏览器向万维网服务器发送这个超链接请求时，服务器检查到这个 URL 指向目录/cgi-bin 下的一个 CGI 脚本 cgiprog。服务器把 HTTP 请求报文头部的信息放在环境变量中，并启动 cgiprog。cgiprog 从环境变量中得到参数并运行，运行的结果送给服务器。服务器形成 HTTP 响应报文发回给浏览器。浏览器显示这一动态文档的页面。

CGI 的名称中出现“网关”一词是因为 CGI 脚本还可以访问其他的应用服务器资源，如数据库等，此时 CGI 脚本的作用像一个网关。前面提到动态文档可用来报告股市行情等，动态的数据经常放在数据库中。CGI 脚本一个经常的应用是作为网关访问数据库。CGI 脚本从数据库中取得动态的数据交给万维网服务器，服务器形成 HTTP 响应报文发回给浏览器，浏览器显示这一动态文档的页面。于是，通过 CGI 形成了 Browser/Web Server/DBMS 这样的应用形成，如图 4-23 所示。CGI 的名称中的“接口”是因为有一些已定义好的变量和调用可供其他 CGI 程序使用。请注意：在看到 CGI 这个名词时，应弄清楚是指 CGI 程序，还是指 CGI 标准。CGI 程序是脚本语言程序，而 CGI 标准是 CGI 程序与万维网服务器交互的机制。

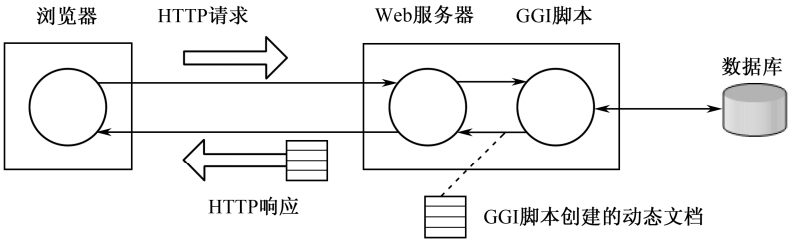


图 4-23　通过 CGI 形成 Browser/Web Server/DBMS 应用

CGI 脚本输出的报文首部应该具有下述格式。

第一行指明文档类型，即 Content-Type：text/html。CGI 脚本的输出不限于 HTML，CGI 标准允许产生其他格式的文档，如普通文本文件和数字图像等，因此需要指明文档类型。

第二行必须是空行，从第三行起才是 HTML 文档。

服务器确认 CGI 脚本送来的报文首部正确无误后，就加上 HTTP 首部的有关信息，将 HTML 文档发给浏览器。

下面是用 C 语言编写的一个 CGI 脚本的示例。

```
        main（）
{   printf（“Content-Type:   text/html\\n\\n”）;
 printf（“<html>”）;
 printf（“The current time is: ”）;
 system（“date”）;
printf（“</html>”）;
}
```

程序运行后，CGI 的输出结果：

```
Content-Type:   text/html
<html>The current time is：March 15 20: 36: 18 2010</html>
```

服务器收到 CGI 脚本的输出结果后，在这个输出结果的前面加上必要的 HTTP 首部，向发起请求的浏览器发送。最后在浏览器的屏幕上显示的内容如下。

```
The current time is：March 15 20: 36: 18 2010
```

时间是随着服务器运行的不同时刻而变化的，浏览器显示的时间是 CGI 脚本运行时刻服务器的时钟。

3．表单

表单（Form）用来将用户数据从浏览器传递给服务器，这在创建动态文档时是很有用的。表单和 CGI 程序经常配合使用，用来创建动态文档。

表单在浏览器的屏幕出现时，可以有一些选择框和按钮，供用户选择和单击。有的方框可让用户录入数据，这样浏览器就可以收集不同用户的不同数据，传递给服务器。

在 HTML 文档的主体中使用表单标签<FORM>和</FORM>来定义一个表单。在<FORM>和</FORM>中间要插入一些标签，来指明表单中所包含项目的细节。在<FORM>标签中首先要说明一个 ACTION 参数，ACTION 参数后面的引号中指出在万维网服务器中 CGI 脚本的位置，一般就是一个 URL。

从浏览器向服务器上的 CGI 脚本发送的一般是用户输入的数据，CGI 脚本负责解释和处理这些数据。例如，这些数据可以写入一个有关的数据库中，供浏览器浏览和查询。

4．PHP、JSP 和 ASP

CGI 还有一些替换技术用于生成动态页面，它们能够处理表单，能够与服务器上的数据库进行交互，可以接收来自表单的信息，在数据库中查找信息，然后利用这些信息生成动态 HTML 页面。

一种替换技术称为超文本预处理器。生成动态 HTML 页面使用的 PHP 脚本，可以放在

<?PHP…？>HTML 标签内。服务器要求包含 PHP 的 Web 页面文件的扩展名为.php，而不是.html/.htm。在支持 PHP 的 Web 服务器上，运行 PHP 脚本就创建了动态 Web 文档。

另一种替换技术是 Sun 公司开发的 Java 服务器页面（Java Server Pages，JSP），它与 PHP 相似，但生成动态 HTML 页面的部分要使用 Java 语言编写，而不是用 PHP 编写，页面文件的扩展名为.jsp。

还有一种替换技术是 Microsoft 公司开发的活动服务器页面（Active Server Pages，ASP）技术，它使用 Microsoft 的脚本语言 Visual Basic Script 来生成动态 HTML 页面内容，页面文件的扩展名为.asp。

4.6.6 活动 Web 文档技术

1．活动文档的创建

动态文档虽然可以在浏览器上显示动态信息，但它使显示屏幕连续刷新的能力是有限的。像动画一类的显示页面需要屏幕连续、快速地刷新，动态文档就无能为力了。这是因为动态文档的屏幕刷新依赖于下述一系列操作：浏览器的请求激活服务器端 CGI 脚本，CGI 脚本的运行更新了动态文档，更新的动态文档返回浏览器。

活动文档技术能满足屏幕连续、快速的刷新。其基本做法：每当浏览器请求一个活动文档，服务器的响应中就返回一段程序，该程序在浏览器上运行，进行屏幕刷新。屏幕刷新工作由浏览器在本地实现，不需要在远程服务器上不断进行文档更新和网络传送，因而可以加快刷新速度。

为了进一步提高运行效率，活动文档可以用压缩形成存储和传送。由于服务器上的活动文档的内容是不变的（这与动态文档技术不同），浏览器还可以在本地缓存一份活动文档的副本。另外，活动文档也不需要包括其运行所需要的全部软件，大部分的支持软件事先存放在浏览器中，无须存于服务器并经网络传送。图 4-24 给出了活动文档的创建过程。

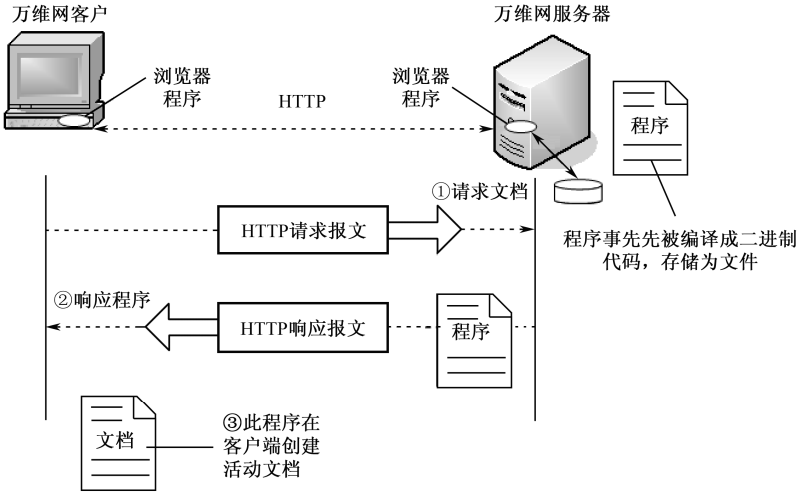


图 4-24　活动文档的创建过程

2．用 Java 创建活动文档

Java 是一种可创建和运行活动文档的技术，它由美国 Sun Microsystem 公司开发。现在它定位于万维网，独立于计算机硬件系统。Java 技术主要由以下 3 部分组成。

1）程序设计语言。Java 程序设计语言简称 Java，它是一种面向对象（Object-oriented）的高级语言，是从 C++派生出来的，保留了许多 C++的语法和语义结构，但它们并不兼容。

2）运行环境。Java 系统定义了运行 Java 程序所必要的运行环境，其中包括一个 Java 虚拟机（Java Virtual Machine，JVM）。一般的编译程序是将源程序编译成特定的计算机系统直接运行的二进制目标程序，但 Java 编译程序则是将源程序转换成一种独立于机器的二进制代码，称为 Java 字节码（Java Bytecode），它被解释执行，在任何计算机上运行都产生同样的输出。

3）Java 类库（Class Library）。为了更容易编写小应用程序，Java 提供了功能强大的类库，定义了几十个类，包括图形操作、网络 I/O、万维网服务器交互、系统访问、文件 I/O 等。

Java 使用小应用程序 Applet 来描述活动文档程序。程序员用 Java 编写一段源程序，然后用编译器编译为字节码形成，就创建了一个 Applet。Java Applet 作为活动文档程序存放于 Web 服务器。

有两种方法可以调用 Java Applet：一种是用户向浏览器进行某个 Applet 的 URL 调用；另一种是 HTML 文档用标签（APPLET）调用。为了将 Applet 嵌入到 HTML 文档中，定义了新的 HTML 标签<APPLET>，它可以指定一个 Applet 的位置。当浏览器看到<APPLET>标签时，就根据标签后面指定的某一 Applet 的位置，与相关的 Web 服务器联系，取来这个 Applet 的副本并解释执行。浏览器应该支持 Java 运行环境。

Java 技术也有一些替代技术，其中一种是 JavaScript。JavaScript 是一种解释性语言，即脚本语言。JavaScript 不把小应用程序编译成字节码形成，而是让浏览器以源代码形式读取和解释脚本。JavaScript 能与 HTML 结合在一起，HTML 页面可以包含 JavaScript 语句。JavaScript 脚本不需要编译，直接嵌入到页面中。在一个 HTML 文档中，标签<SCRIPT>用来插入一段 JavaScript 的语句。JavaScript 简单易用，但速度比 Java Applet 慢，传送和解释源代码都要比字节码更花费时间。

Microsoft 的活动文档技术是指 Web 页面中包含 ActiveX 控件（ActiveX Control）。ActiveX 控件是一种已经被编译成 Pentium 机器指令的程序，可以直接在硬件上执行，比解释执行的 Java Applet 更快、更灵活。当 IE 浏览器在 Web 页面中发现一个 ActiveX 控件时，它会下载这个控件，然后执行它。

小结

本章讨论了位于网络结构中的最高层——应用层，其协议与底层协议之间有着千丝万缕的联系，这些协议之间的相互作用是通过 C/S 模式来进行的。而构成应用层协议的就是 InternetDNS、文件传输协议、电子邮件协议和万维网等。因此可以从本章这几大类传输协议上

进行掌握。

首先，对于 Internet DNS，它是用于实现网络设备名字到 IP 地址映射的网络服务，它和 IP 是一一对应的。域名是一个逻辑概念，并不一定与物理地点相一致。它的各层级的划分与地址映射也是需要读者掌握并记忆的。其次，对于文件传输协议（FTP），它用于实现 Internet 中交互文件传输功能。本章详细列举了它的构建模型及各模型的含义和作用，读者需细心钻研。再次，电子邮件协议用于实现 Internet 中电子邮件的传送功能。它作为使用最多、最受用户青睐的协议，也发挥了其特有的通信机制。最后，万维网也称环球信息网，是 Internet 的一种应用系统。它本质上通过链接的方式实现 Internet 上一个 Web 站点到另一个站点的访问。

第 5 章 网络安全

网络安全是指网络系统的硬件、软件及其系统中的数据受到保护，不因偶然的或者恶意的原因而遭受到破坏、更改、泄露，系统连续、可靠、正常地运行，网络服务不中断。网络安全从其本质上来讲就是网络上的信息安全。本章主要讲述网络安全中的防火墙、信息加密以及网络或计算机病毒。

5.1 防火墙

5.1.1 防火墙简介

为了保护网络（特别是企业内部网）资源的安全，人们创建了网络防火墙。就像建筑物防火墙或护城河能够保护建筑物及其内部资源安全或保护城市免受侵害一样，网络防火墙能够防止外部网上的各种危害侵入内部网。

目前，防火墙已在 Internet 上得到了广泛的发展，且由于具有不限于 TCP/IP 的特点，也使其逐步在 Internet 之外更具生命力。客观地讲，防火墙并不是解决网络安全问题的万能药方，只是网络安全政策和策略中的一个组成部分。但了解防火墙技术并学会在实际操作中应用防火墙技术，相信会在“网络经济”社会的工作和生活中使每一位用户都受益匪浅。

1．防火墙的概念和功能

（1）防火墙的概念

防火墙的本义是指古代人们房屋之间修建的一道墙，这道墙可以防止火灾发生时蔓延到其他房屋。而这里所说的网络防火墙当然不是指物理上的防火墙，而是指隔离在本地网络与外部网络之间的一道防御系统。应该说，在互联网上防火墙是一种非常有效的网络安全措施，通过它可以隔离风险区域（Internet 或有一定风险的网络）与安全区域（区域内部网，也可称为可信任网络）的连接，同时不会妨碍人们对风险区域的访问。

网络防火墙就是指在企业内部网和外部网之间所设立的执行访问控制策略的安全系统。它在内部网和 Internet 之间设置控制，以防止发生不可预测的、外界对内部网资源的非法访问或潜在破坏性的侵入。它是目前实现网络安全策略的最有效的工具之一，也是控制外部用户访问内部网的第一道关口。防火墙的设置思想就是在内部、外部两个网络之间建立一个具有安全控制机制的安全控制点，通过允许、拒绝或重新定向经过防火墙的数据流，来实现对内部网服务和访问的安全审计和控制。需要指出的是，防火墙虽然可以在一定程度上保护内部网的安全，但内部网还应该有其他的安全保护措施，这是防火墙所不能代替的。

防火墙技术是建立在现代通信网络技术和信息安全技术基础上的应用性安全技术，越来越多地被应用于专用网络与公用网络的互连环境中，尤其以接入 Internet 网络最普遍。防火墙可通过监测、控制跨越防火墙的数据流，尽可能地对外界屏蔽内部网络的信息、结构和运行状态，以此来实现内部网络的安全保护。

防火墙可由计算机硬件和软件系统组成。通常情况下，内部网和外部网进行互连时，必须使用一个中间设备，这个设备既可以是专门的互连设备（如路由器或网关），也可以是网络中的某个节点（如一台主机）。这个设备至少具有两条物理链路，一条通往外部网络，一条通往内部网络。企业用户希望与其他用户通信时，信息必须经过该设备，同样，其他用户希望访问企业网时，也必须通过该设备。显然，该设备是阻挡攻击者入侵的关口，也是防火墙实施的理想位置，如图 5-1 所示。

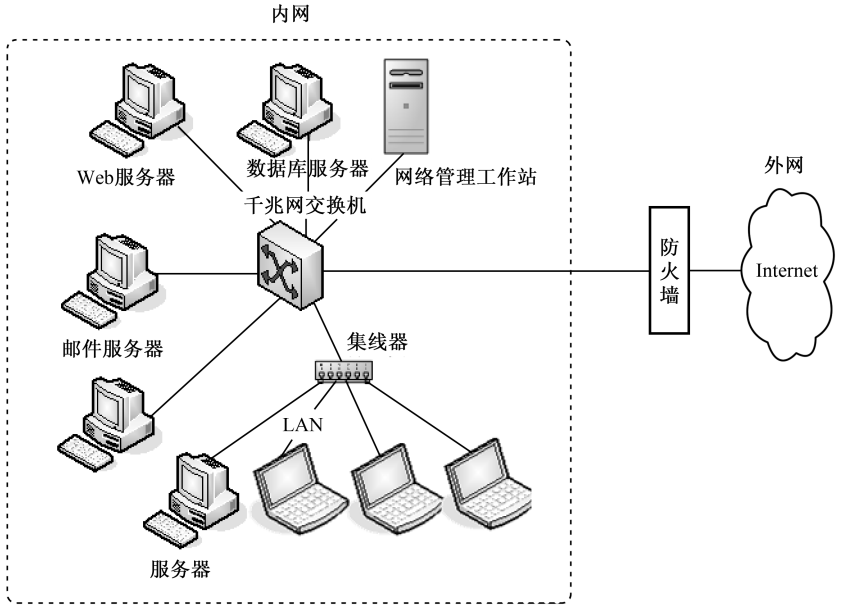


图 5-1　防火墙的位置示意图

防火墙的作用是防止不希望的、未授权的通信进出被保护的网络，使机构强化自己的网络安全政策。由于防火墙设定了网络边界和服务，因此更适合于相对独立的网络，如 Internet。事实上，在 Internet 上的 Web 网站中，超过 1/3 的 Web 网站都是用某种形式的防火墙加以保护的。

可以说，防火墙能够限制非法用户从一个被严格保护的设备上进入或离开，从而有效地阻止对内部网的非法入侵。但由于防火墙只能对跨越边界的信息进行检测、控制，而对网络内部人员的攻击不具备防范能力，因此单独依靠防火墙来保护内部网络的安全是不够的，还必须与入侵检测系统（IDS）、安全扫描、应急处理等其他安全措施综合使用才能达到目的。

（2）防火墙的功能

一般来说，防火墙在配置上可防止来自“外部”未经授权的交互式登录，这大大有助于防止破坏者登录到网络用户的计算机上。一些设计更为精巧的防火墙既可以防止来自外部的信息流进入内部，又允许内部的用户可以自由地与外部通信。如果切断防火墙，就可以保护用户免受网络上任何类型的攻击。

防火墙的另一个非常重要的作用是可以提供一个单独的“阻塞点”，在“阻塞点”上设置安全和审计检查。防火墙可提供一种重要的记录和审计功能，提供有关通过防火墙的数据流的类型和数量，以及有多少次试图闯入防火墙的企图等信息。

利用防火墙保护内部网，主要有以下几个主要功能。

1）网络安全的屏障。防火墙是信息进出网络的必经之路，它可检测所有经过数据的细节，并根据事先定义好的策略允许或禁止这些数据的通过。一个防火墙可极大地提高内部网络的安全性，并通过过滤不安全的服务而降低风险。由于只有经过精心选择的应用协议才能通过防火墙，所以能使网络环境变得更安全。这样外部的攻击者就不可能利用这些脆弱的协议来攻击内部网络。防火墙同时可以保护网络免受基于路由的攻击。

2）强化网络安全策略。通过以防火墙为中心的安全方案配置，能将所有的安全问题（如口令、加密、身份认证、审计等）配置在防火墙上。与将网络安全问题分散到各个主机上相比，防火墙的集中式安全管理更经济。例如，在网络访问时，密钥码系统和其他的身份认证系统完全可以不必分散在各个主机上，而是集中在防火墙上。

3）对网络存取和访问进行监控审计。如果所有的访问都要经过防火墙，那么，防火墙就能记录这些访问并做出日志记录，同时也能提供网络使用情况的统计数据。当发生可疑动作时，防火墙能进行报警，并提供网络是否受到监测和攻击的详细信息。另外，使用统计手段对网络进行需要分析和威胁分析等也是非常重要的。

4）防止内部信息的外泄。通过利用防火墙对内部网络的划分，可实现对内部网重点网段的隔离，从而限制局域重点或敏感的网络安全问题对全局网络造成的影响。在内部网络中，不引人注意的细节可能包含了有关安全的线索，从而引起外部攻击者的兴趣，甚至因此暴露了内部网络的某些安全漏洞，使用防火墙就可以隐藏那些内部细节。防火墙可以同样阻塞有关内部网络的 DNS 信息，这样，一台主机的域名和 IP 地址就不会被外界了解。

5）安全策略检查。所有进出网络的信息都必须通过防火墙，防火墙成为网络上的一个安全检查站，对外部的网络进行检测和报警，将检查出来的可疑的访问拒之网外。

（3）防火墙不能做什么

防火墙可使内部网在很大程度上免受攻击，但认为配置了防火墙，所有的网络安全问题都迎刃而解的想法是错误的，起码是不全面的。可以说许多危险是防火墙所无能为力的，即防火墙还存在一些不足之处。

1）防火墙不能防范内部人员的攻击。防火墙只能提供周边防护，并不能控制内部用户对内部网络滥用授权的访问。内部用户可窃取数据、破坏硬件和软件，并可巧妙地修改程序而不接近防火墙。内部用户攻击网络正是网络安全最大的威胁。统计表明，很多安全事件是由于内部人员的攻击造成的，由内部引起的安全问题约占总数的 80%。

2）防火墙不能防范绕过它的连接。防火墙可有效地检查经过它进行传输的信息，但不能防止绕过它传输的信息。例如，如果站点允许防火墙后面的内部系统进行拨号访问，那么防火墙就没有办法阻止攻击者进行的拨号入侵。

3）防火墙不能防御全部威胁。防火墙可防御已知的威胁，如果是一个很好的防火墙设计方案，可以防御新的威胁，但没有一个防火墙能够防御所有的威胁。

4）防火墙不能防御恶意程序和病毒。虽然许多防火墙能扫描所有通过的信息，以决定是否允许它们通过防火墙进入内部网络，但扫描是针对源、目的地址和端口号的，而不扫描数据的确切内容。因为在网络上传输的二进制文件的编码方式很多，并且有太多的不同结构的病毒，因此防火墙不可能查找到所有的病毒，也就不能有效地防范像病毒这类程序的入侵。如今恶意程序发展迅速，病毒可依附于毒源，许多站点都可以下载病毒程序甚至源码。某些防火墙可以根据已知病毒和木马的特征码检查数据流，虽然这样做有些帮助但并不可靠，因为类似的恶意程序的种类很多，有多种手段可使它们在数据中隐藏，防火墙对那些新的病毒和木马程序等是无能为力的。此外，防火墙只能发现从其他网络来的恶意程序，但许多病毒却是通过被感染的软盘或系统直接进入网络的。所以，对病毒等恶意程序十分敏感的单位应当在整个机构范围内采取病毒控制措施。

2．个人防火墙

现在网上流行很多个人防火墙软件，它是应用程序级的。个人防火墙是一种能够保护个人计算机系统安全的软件，是可以直接在用户计算机操作系统上运行的软件服务。通常，这些防火墙安装在计算机网络接口的较低级别上，使它们可以监视通过网卡的所有网络通信。

一旦安装上个人防火墙，就可以把它设置成“学习模式”，这样，对遇到的每一种新的网络通信，个人防火墙都会提示用户一次，询问如何处理这种通信。然后，个人防火墙便记住了其响应方式，并应用于以后遇到的同种网络通信。例如，如果用户已经安装了一台个人 Web 服务器，个人防火墙可能对第一个传入的 Web 连接做一标记，并询问用户是否允许它通过。用户可以允许所有的 Web 连接、来自某些特定 IP 地址范围的连接等，个人防火墙就将这些规则应用于此后所有传入的 Web 连接。

可以将个人防火墙想象成为用户计算机上建立的一个虚拟网络接口，不再是计算机操作系统直接通过网卡进行通信，而是操作系统与个人防火墙的对话，仔细检查网络通信，然后再通过网卡通信。

（1）个人防火墙的优点

1）增加了保护功能。个人防火墙具有安全保护功能，既可以抵挡外来攻击，又可以抵挡内部的攻击。例如，家庭用户使用调制解调器或 ISDN/ADSL 上网，个人防火墙就能够为用户隐藏暴露在网络上的信息（如 IP 地址）。

2）易于配置。个人防火墙产品通常可以使用直接的配置选项获得基本可使用的配置。

3）廉价。个人防火墙不需要额外的硬件资源就为内部网的个人用户提供廉价的网络保护。

（2）个人防火墙的缺点

1）接口通信受限。个人防火墙对公共网络只有一个物理接口，而真正的防火墙应当控制两个或更多的网络接口之间的通信。因此，个人防火墙本身可能会容易受到威胁，或者说具有网络通信可以绕过防火墙的规则这样的弱点。

2）集中管理比较困难。个人防火墙需要在每个客户端进行配置，这将增加管理开销。

3）性能受限。个人防火墙是为了保护单个计算机系统而设计的，但是如果安装它的计算机是与内部网络上的其他计算机共享到 Internet 的连接，则它也可以保护小型网络。个人防火墙在充当小型网络路由器时将导致性能下降。这种保护机制通常不如专用防火墙方案有效，因为它们通常只限于阻止 IP 和端口地址。

3．内部防火墙

防火墙主要保护内部网络资源免受外部用户的非法访问和侵袭。为了保护内部重要信息的安全，有时也需要对内部网的部分站点再加以保护，以免受内部网其他站点的侵袭。因此，需要在同一结构的两个部分之间，或者在同一内部的两个不同组织结构之间再建立一层防火墙，这就是内部防火墙。

企业内部网是一种多层次、多节点、多业务的网络，各节点间的信任程度较低，但由于业务的需要，各节点和服务器之间又要频繁地交换数据。通过在服务器群的入口处设置内部防火墙，可有效地控制内部网络的访问。企业内部网中设置内部防火墙后，一方面可以有效地防范来自外部网络的攻击，另一方面可以为内部网络制定完善的安全访问策略，从而使得整个企业网络具有较高的安全级别。

内部防火墙的用户包括内部网本单位的雇员（如内部网单位本部的用户、本单位外部的用户、本单位的远程用户或在家中办公的用户）和单位的业务合作伙伴。后者的信任级别比前者低。

许多用于建立外部防火墙的工具与技术也可以用于建立内部防火墙。

内部防火墙可以实现以下功能。

1）精确地制定每个用户的访问权，保证内部网络用户只能访问必要的资源。

2）对于拨号备份路线的连接，通过强大的认证功能，实现对远程用户的管理。

3）记录网段间的访问信息，及时发现误操作和来自内部网络其他网段的攻击行为。

4）通过安全策略的集中管理，每个网段上的主机不必再单独设立安全策略，降低人为因素导致的网络安全问题。

5.1.2　防火墙技术

1．防火墙类型

根据防火墙的分类标准不同，可将防火墙分为不同的类型。

（1）基于防火墙技术原理分类

Internet 采用 TCP/IP，在不同的网络层次上设置不同的屏障，构成不同类型的防火墙。因此，从工作原理角度看，防火墙技术主要可分为网络层防火墙技术和应用层防火墙技术。这两个层次的防火墙技术的具体实现有包过滤防火墙、代理服务器防火墙、状态检测防火墙和自适应代理防火墙。

（2）基于防火墙硬件环境分类

根据实现防火墙的硬件环境的不同，可将防火墙分为基于路由器的防火墙和基于主机系统的防火墙。包过滤防火墙和状态检测防火墙可以基于路由器，也可以基于主机系统实现；而代理服务器防火墙只能基于主机系统实现。

（3）基于防火墙的功能分类

根据防火墙功能的不同，可将防火墙分为 FTP 防火墙、Telnet 防火墙、E-mail 防火墙、病毒防火墙、个人防火墙等专用防火墙。通常也将几种防火墙技术组合在一起使用以弥补各自的缺陷，增加系统的安全性能。

2．包过滤技术

（1）包过滤技术的工作原理

网络层防火墙技术根据网络层和传输层的原则对传输的信息进行过滤。网络层技术的一个范例就是包过滤（Packet Filtering）技术。因此，利用包过滤技术在网络层实现的防火墙又称包过滤防火墙。在基于 TCP/IP 的网络上，所有往来的信息都被分割成许许多多一定长度的数据包，包中包含发送者的 IP 地址和接收者的 IP 地址等信息。当这些数据包被送入互联网时，路由器会读取接收者的 IP 地址信息并选择一条合适的物理线路发送数据包。数据包可能经由不同的路线到达目的地，当所有的包到达目的地后会重新组装还原。

包过滤技术是在网络的出入口（如路由器）对通过的数据包进行检查和选择的。选择的依据是系统内设置的过滤逻辑（包过滤规则），也称为访问控制表（Access Control Table）。通过检查数据流中每个数据包的源地址、目的地址、所有的端口号、协议状态或它们的组合，来确定是否允许数据包通过。通过检查，只有满足条件的数据包才允许通过，否则被抛弃（过滤）。如果防火墙中设定某一 IP 地址的站点为不适宜访问的站点，则从该站点地址来的所有信息都会被防火墙过滤。这样可以有效地防止恶意用户利用不安全的服务器对内部网进行攻击。包过滤防火墙要遵循的一条基本原则就是“最小特权原则”，即明确允许管理员希望通过的那些数据包，禁止其他数据包通过。

在网络上传输的每个数据包都可分为数据和包头两部分。包过滤器就是根据包头信息来判断该包是否符合网络管理员设定的规则表中的规则，以确定是否允许数据包通过。包过滤规则一般是基于部分或全部包头信息的，如 IP 类型、IP 源地址、IP 选择域的内容、TCP 源端口号、TCP 目标端口号等。例如，包过滤防火墙可以对来自特定的 Internet 地址的信息进行过滤，或者只允许来自特定地址的信息通过。它还可以根据需要的 TCP 端口来过滤信息。如果将过滤设置成只允许数据包通过 TCP 端口 80（标准的 HTTP 端口），那么在其他端口[如端口 25（标准的 SMTP 端口）]上的服务程序的数据包均不得通过。

包过滤防火墙既可以允许授权的服务程序和主机直接访问内部网络，又可以过滤指定的端

口和内部用户的 Internet 地址信息。大多数包过滤防火墙的功能可以设置在内部网络与外部网络之间的路由器上，作为第一道安全防线。路由器是内部网络与 Internet 连接必不可少的设备，因此在原有网络上增加这样的防火墙软件几乎不需要任何额外的费用。

（2）过滤路由器与普通路由器

增加了包过滤防火墙软件、具备了过滤特性的路由器称为过滤路由器。

普通路由器只简单地查看每一数据包的目的地址，并选择数据包发往目标地址的最佳路径。当路由器知道如何发送数据包到目标地址时，则发送该包；如果不知道如何发送数据包到目标地址，则返还数据包，并通知源地址“数据包不能到达目标地址”。在对数据包做出路由决定时，普通路由器只依据包的目的地址引导包，而过滤路由器将更严格地检查数据包，除了决定它是否发送数据包到达其目标外，过滤路由器还决定数据是否应该发送。“应该”或“不应该”由站点的安全策略决定，并由过滤路由器强制执行。过滤路由器依据路由器中的包过滤规则做出是否引导该包的决定。过滤路由器以包的目标地址、包的源地址和包的传输协议为依据，确定允许或不允许某些包在网上传输。

（3）包过滤规则

包过滤防火墙的过滤规则的主要描述形式有逻辑过滤表、文件过滤规则表和内存过滤表。在包过滤系统中，规则表是十分重要的。依据规则表可检查过滤模块、端口映射模块和地址欺骗等。规则表制定的好坏，直接影响机构的安全策略是否会被有效地体现；规则表设置的结构是否合理，将影响包过滤防火墙的性能。

通常，包过滤技术允许或不允许某些数据包通过，主要依据包的目的地址、包的源地址和包的传输协议。大多数包过滤系统判决是否传输包都不关心包的具体内容，而是让用户进行如下操作。

- 不允许任何用户从外部网络用 Telnet 登录。
- 允许任何用户使用 SMTP 往内部网络发送电子邮件。
- 只允许某台机器通过 NNTP（网络新闻传输协议）往内部网络发送新闻。

（4）包过滤防火墙的特点

1）包过滤技术的优点。

- 一个过滤路由器能协助保护整个网络。数据包过滤的主要优点之一就是一个恰当放置的包过滤路由器有助于保护整个网络。如果仅有一个路由器连接内部与外部网络，不论内部网络的大小和内部拓扑结构如何，通过该路由器进行数据包过滤，就可以在网络安全保护上取得较好的效果。
- 包过滤对用户透明。数据包过滤不要求任何自定义软件或客户机配置，也不要求对用户进行任何特殊的训练或操作。当包过滤路由器决定让数据包通过时，它与普通路由器没有区别。比较理想的情况是用户没有感觉到它的存在，除非它们试图做过滤规则中所禁止的事。较强的“透明度”是包过滤的一大优势。
- 过滤路由器速度快、效率高。过滤路由器只检查包头相应的字段，一般不查看数据包的内容，而且某些核心部分是由专用硬件实现的，故其转发速度快、效率高、技术通用、价廉、有效。包过滤技术不是针对各个具体的网络服务采取特殊的处理方式，而

是对各种网络服务都通用，大多数路由器都提供包过滤功能，不用再增加更多的硬件和软件，因此其价格低廉，能很大程度地满足企业的安全要求，其应用行之有效。

2）包过滤技术的缺点。

- 安全性较差。防火墙过滤的只有网络层和传输层的有限信息，因而各种安全要求不可能充分满足；在许多过滤器中，过滤规则的数目是有限的，且随着规则数目的增加，性能将受到影响。过滤路由器只检测 TCP/IP 包头，检查特定的几个域，而不是检查数据包的内容，不按特定的应用协议进行审查和扫描，不做详细分析和记录。非法访问一旦突破防火墙，即可对主机上的软件和配置漏洞进行攻击。因而，与代理技术相比，包过滤技术的安全性较差。
- 不能彻底防止地址欺骗。大多数包过滤路由器都是基于源 IP 地址、目的 IP 地址而进行过滤的。而 IP 地址的伪造是很容易、很普遍的。如果攻击者将自己主机的 IP 地址设置成为一个合法主机的 IP 地址，就可以轻易地通过路由器。因此，过滤路由器在 IP 地址欺骗上大都无能为力，即使按 MAC 地址进行绑定，也是不可信的。因此，对于一些安全性要求较高的网络，过滤路由器是不能胜任的。
- 一些应用协议不适合于数据包过滤。即使是完美的数据包过滤实现，也会发现一些协议不太适合于数据包过滤安全保护，如 RPC、X-Window 和 FTP。
- 无法执行某些安全策略。包过滤路由器上的信息不能完全满足人们对安全策略的需要。例如，数据包表明它们来自什么主机，而不是什么用户，因此，我们不能强制限制特殊的用户。同样，数据包表明它到什么端口，而不是到什么应用程序。当我们通过端口号对高级协议强行限制时，不希望在端口上有指定协议之外的其他协议，恶意知情者能够容易地破坏这种控制。

从以上分析可以看出，包过滤技术虽然能够确保一定的安全保护，且也有许多优点，但是它毕竟是早期的防火墙技术，本身存在较多缺陷，不能提供较高的安全性。在实际应用中，很少把这种技术作为单独的安全解决方案，而是把它与其他防火墙技术组合在一起使用。

3．代理服务技术

（1）代理服务技术的工作原理

代理服务器防火墙工作在 OSI 模型的应用层，它掌握着应用系统中可用作安全决策的全部信息，因此，代理服务器防火墙又称应用层网关。这种防火墙通过一种代理（Proxy）技术参与到一个 TCP 连接的全过程。从内部网用户发出的数据包经过这样的防火墙处理后，就好像是源于防火墙外部网卡一样，从而可以达到隐藏内部结构的作用。代理服务技术通过主机上运行代理的服务程序，直接对特定的应用层进行服务，因此也称为应用层防火墙，其核心是运行于防火墙主机上的代理服务器进程。

代理服务器是代表客户处理在服务器连接请求的程序。当代理服务器得到一个客户的连接意图时，对客户的请求进行核实，并经过特定的安全化代理应用程序处理连接请求，将处理后的请求传递到真正的 Internet 服务器上，然后接收服务器应答。代理服务器对真正服务器的应答做进一步处理后，将答复交给发出请求的最终客户。代理服务器通常运行在两个网络之间，它

对于客户来说像是一台真正服务器，而对于外部网的服务器来说，它又似一台客户机。代理服务器并非将用户的全部网络请求都提交给 Internet 上的真正服务器，而是先依据安全规则和用户的请求做出判断，是否代理执行该请求，一些请求可能被否决。当用户提供了正确的用户身份及认证信息后，代理服务器建立与外部 Internet 服务器的连接，为两个通信点充当中继。内部网络只接收代理服务器提出的要求，拒绝外部网的直接请求。代理服务器的工作原理示意图如图 5-2 所示。

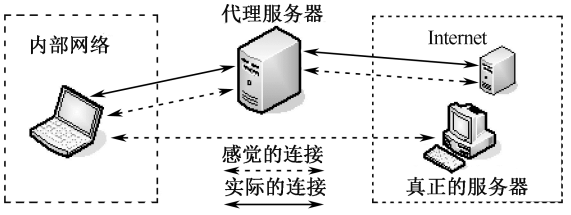


图 5-2　代理服务器的工作原理示意图

一个代理服务器本质上就是一个应用层网关，即一个为特定网络应用而连接两个网络的网关。代理服务器像一堵墙一样挡在内部用户和外界之间，分别与内部和外部系统连接，是内部网与外部网的隔离点，起着监视和隔绝应用层通信流的作用。从外部只能看到该代理服务器而无法获知任何的内部资源，如用户的 IP 地址等。

代理服务器技术能够记录一些通过它的信息，如什么用户在什么时间访问过什么站点等。这些信息可以帮助网络管理员识别网络间谍。代理服务器通常都拥有一个高速 Cache，该 Cache 存储用户频繁访问的站点内容（页面），在下一个用户要访问该站点的这些内容时，代理服务器就不用连接到 Internet 上的服务器重复地获取相同的内容，而是直接将本身 Cache 中的内容发出即可，从而节约了访问的响应时间和网络资源。

许多代理服务器防火墙除了提供代理业务外，还提供网络层的信息过滤功能。它们也对过往的数据包进行分析和注册登记，形成报告，同时当发现被攻击迹象时会向网络管理员发出警报，并保留攻击痕迹。

代理服务可以实现用户认证、详细日志、审计跟踪和数据加密等功能，并实现对具体协议及应用过滤，如阻塞 Java 或 JavaScript。代理服务器技术能完全控制网络信息的交换，控制会话过程，具有灵活性和安全性，但可能影响网络的性能，对用户不透明，且对每一种服务器都要设计一个代理模块，建立对应的网关层，实现起来比较复杂。

（2）代理服务器的实现

1）代理服务器。

应用代理服务可以在网络应用层提供授权检查及代理服务功能。当外部某台主机试图访问受保护的内部网时，它必须先在防火墙上经过身份认证。通过身份认证之后，防火墙运行一个专门程序，把外部主机与内部主机连接。在这个过程中，防火墙可以限制用户访问的主机、访问时间及访问方式。同样，受保护的内部网络用户访问外部网时也需先登录到防火墙上，通过验证后才可使用 Telnet 或 FTP 等有效命令。应用代理服务器的优点是既可以隐藏内部的 IP 地址，又可以给单个用户授权。即使攻击者盗用了一个合法的 IP 地址，它也要通过严格的身份认证。但是这种认证使得应用网关不透明，用户每次连接都要受到“盘问”，这会给用户带来许多不便。而且这种代理服务技术需要为每个应用网关编写专门的程序。

2）回路级代理服务器。

回路级代理服务器也称一般代理服务器，它适用于多个协议，但不解释应用协议中的命令就建立了连接回路。回路级代理服务器通常要求使用修改过的用户程序。套接字服务器（Sockets Server）就是回路级代理服务器。套接字（Sockets）是一种网络应用层的国际标准。当受保护的网络客户机需要与外部网交互信息时，在防火墙上的套接字服务器检查客户的 User ID、IP 源地址和 IP 目的地址，经过确认后，套接字服务器才与外部服务器建立连接。对用户来说，受保护的内部网与外部网的信息交换是透明的，感觉不到防火墙的存在，这是因为 Internet 用户不需要登录到防火墙上。

回路级代理服务器可为各种不同的协议提供服务。大多数回路级代理服务器也是公共服务器，它们几乎支持任何协议，但不是每个协议都能由回路级代理服务器轻易实现。

3）智能代理服务器。

如果一个代理服务器不仅能处理转发请求，同时还能够做许多其他事情，这种代理服务器称为智能代理服务器。智能代理服务器可提供比其他方式更好的日志和访问控制功能。一个专用的应用代理服务器很容易升级到智能代理服务器，而回路级代理服务器则较困难。

4）邮件转发服务器。

当防火墙采用相应技术使得外部网络只知道防火墙的 IP 地址和域名时，从外部网络发来的邮件就只能送到防火墙上。这时防火墙对邮件进行检查，只有发送邮件的源文件是被允许通过的。防火墙对邮件的目的地址进行转换，送到内部的邮件服务器，由其进行转发。

（3）代理服务器技术的特点

1）代理服务器技术的优点。

- 安全性好。由于每一个内、外网络之间的连接都要通过代理服务技术的介入和转换，通过专门为特定的服务（如 http）编写的安全化应用程序进行处理，然后由防火墙本身分别向外部服务器提交请求和向内部用户发回应答，没有给内、外网络的计算机以任何直接会话的机会，从而避免了入侵者使用数据驱动类型的攻击方式入侵内部网。另外，代理服务技术还按特定的应用协议对数据包内容进行审查和扫描，因此也增加了防火墙的安全性。安全性好是代理服务器技术的突出优点。
- 易于配置。因为代理服务是一个软件，所以它较过滤路由器更易配置，配置界面十分友好。如果代理服务器实现得好，可以降低配置协议的要求，从而避免配置错误。
- 能生成各项记录。代理服务技术工作在应用层，可检查各项数据，所以可以按一定准则，让代理生成各项日志和记录。这些日志和记录对于流量分析、安全检验是十分重要的。
- 能灵活、完全地控制进出的流量和内容。通过采取一定的措施，按照一定的规则，借助于代理技术实现一整套的安全策略，如控制“谁”和“做什么”，在什么“时间”和“地点”控制等。
- 能过滤数据内容。可以把一些过滤规则应用于代理，让它在高层实现过滤功能，如文本过滤、图像过滤、预防病毒或扫描病毒等。
- 能为用户提供透明的加密机制。用户通过代理服务收发数据，可以让代理服务完成加/解密功能，从而方便用户，确保数据的保密性。这点在 VPN 中特别重要。代理服务

可以广泛地用于企业内部网中，提供较高安全性的数据通信。

- 可以方便地与其他安全技术集成。目前的安全问题解决方案很多，如验证（Authentication）、授权（Authorization）、账号（Accounting）、数据加密、安全协议（SSL）等。如果把代理与这些技术联合使用，将大大增加网络的安全性。

2）代理服务器技术的缺点。

- 速度较慢。代理服务技术工作在应用层，要检查数据包的内容，按特定的应用协议（如HTTP）审查、扫描数据包内容，并进行代理服务，故其速度较慢。
- 对用户不透明。许多代理要求客户端做相应改动或安装定制客户软件，这给用户增加了不透明度。
- 对于不同的服务代理，可能要求不同的服务器。可能需要为每项协议设置一个不同的代理服务器，因为代理服务器不得不理解协议，以便判断什么是允许的和不允许的，并且还要装扮成一个对真实服务器来说它就是客户、对客户来说它就是服务器的角色。选择、安装和配置所有这些不同的服务器是一项较繁重的工作。
- 通常要求对客户或过程进行限制。除了一些为代理而设置的服务，代理服务器要求对客户或过程进行限制，每一种限制都有不足之处，人们无法经常按他们自己的步骤使用快捷可用的方式。由于这些限制，代理应用层不能像非代理应用运行得那样好，它们往往可能曲解协议的说明。
- 代理不能改进底层协议的安全性。因为代理工作于 TCP/IP 的应用层，所以它不具有改善底层通信协议的能力，如 IP 欺骗、SYN 泛滥、伪造 ICMP 消息和一些拒绝服务的攻击。

4．状态检测技术

（1）状态检测技术的工作原理

状态检测技术由 Check Point 率先提出，又称动态包过滤技术。状态检测技术是新一代的防火墙技术。这种技术具有非常好的安全特性。它使用了一个在网关上执行网络安全策略的软件模块，称为检测引擎。检测引擎在不影响网络政策允许的前提下，采用抽取有关数据的方法对网络通信的各层实施检测。它将抽取的状态信息动态地保存起来作为以后执行安全策略的参考。检测引擎维护一个动态的状态信息表并对后续的数据包进行检查。一旦发现任何连接的参数有意外变化，该连接就被终止。

状态检测技术监视和跟踪每一个有效连接的状态，并根据这些信息决定网络数据包是否能够通过防火墙。它在协议栈底截取数据包，然后分析这些数据包，并且将当前数据包和状态信息与前一时刻的数据包和状态信息进行比较，从而得到该数据包的控制信息，来达到保护网络安全的目的。

检测引擎支持多种协议和应用程序，并可以很容易地实现应用和服务的扩充。与前两种防火墙不同，当用户访问请求到达网关的操作系统前，状态监视器要收集有关数据进行分析，结合网络配置和安全规定做出接纳或拒绝、身份认证、报警处理等动作。一旦某个访问违反了安全规定，该访问就会被拒绝，并报告有关状态，做日志记录。

状态检测技术试图跟踪通过防火墙的网络连接和包，这样它就可以使用一组附加的标准，以确定是否允许和拒绝通信。状态检测防火墙是在使用了基本包过滤防火墙的通信基础上应用一些技术来做到这一点的。为了跟踪包的状态，状态检测防火墙不仅跟踪包中包含的信息，还记录有用的信息以帮助识别包。

状态检测技术可检测无连接状态的远程过程调用（RPC）用户数据包（UDP）之类的端口信息，而包过滤和代理服务技术都不支持此类应用。状态检测防火墙无疑是非常坚固的，但它会降低网络的速度，且配置也比较复杂。好在有关防火墙厂家已注意到这个问题。例如，Check Point 公司的防火墙产品 Firewall-1，所有的安全策略规则都是通过面向对象的图形用户界面（GUI）定义的，因此可以简化配置过程。

（2）通过状态检测防火墙的数据包类型

状态检测防火墙在跟踪连接状态方式下通过数据包的类型有 TCP 包和 UDP 包。

- TCP 包。当建立起一个 TCP 连接时，通过的第一个包被标有包的 SYN 标识。通常，防火墙丢弃所有外部的连接企图，除非已经建立起某条特定规则来处理它们。对内部到外部主机的连接，防火墙注明连接包，允许响应两个系统之间的包，直到连接结束为止。在这种方式下，传入的包只有在它响应一个已建立的连接时，才会被允许通过。
- UDP 包。UDP 包比 TCP 包简单，因为它们不包含任何连接或序列信息，只包含源地址、目的地址、检验和携带的数据。这些简单的信息使得防火墙很难确定包的合法性，因为没有打开的连接可利用，以测试传入的包是否被允许通过。但如果防火墙跟踪包的状态，就可以确定。对传入的包，若它所使用的地址和 UDP 包携带的协议与传出的连接请求匹配，该包就被允许通过。

（3）状态检测技术的特点和应用

状态检测技术结合了包过滤技术和代理服务技术的特点。与包过滤技术一样的是它对用户透明，能够在 OSI 网络层上通过 IP 地址和端口号过滤进出的数据包；与代理服务技术一样的是可以在 OSI 应用层上检查数据包内容，查看这些内容是否符合安全规则。

状态检测技术克服了包过滤技术和代理服务技术的局限性，能根据协议、端口及源地址、目的地址的具体情况决定数据包是否通过。对于每个安全策略允许的请求，状态检测技术启动相应的进程，可快速地确认符合授权标准的数据包，使得允许速度加快。

状态检测技术的缺点是状态检测可能造成网络连接的某种迟滞，不过硬件运行速度越快，这个问题就越不易察觉。

状态检测防火墙已经在国内外得到广泛应用，目前在市场上流行的防火墙大多属于状态检测防火墙，因为该防火墙对于用户透明，在 OSI 最高层上加密数据，不需要再去修改客户端程序，也不需对每个需要在防火墙上运行的服务额外增加一个代理。

5．自适应代理技术

新近推出的自适应代理（Adaptive Proxy）防火墙技术，本质上也属于代理服务技术，但它结合了动态包过滤（状态检测）技术。

自适应代理技术是最近商业应用防火墙中实现的一种革命性的技术。组成这类防火墙的基

本要素有两个，即自适应代理服务器与动态包过滤器。它结合了代理服务防火墙的安全性和包过滤防火墙的高速度等优点，在保证安全性的基础上将代理服务器防火墙的性能提高了 10 倍以上。

在自适应代理与动态包过滤器之间存在一种控制通道。在对防火墙进行配置时，用户仅仅将所需要的服务类型、安全级别等信息通过相应代理的管理界面进行设置即可。然后，自适应代理就可以根据用户的配置信息，决定是使用代理服务器从应用层代理请求，还是使用动态包过滤器从网络层转发包。如果是后者，它将动态地通知包过滤器增减过滤规则，满足用户对速度和安全性的双重要求。

5.2　信息加密

5.2.1　信息加密简介

加密是迄今为止在网络和通信安全中最重要的安全技术之一。密码体制经历了从古典密码体制到现代密码体制的发展过程，而现代密码体制有两个方向的分支，即对称密码体制和非对称密码体制。

1．数据加密技术

网络通信的双方称为发送者和接收者。发送者在发送消息给接收者的时候，希望所发送的消息能安全发送到接收者的手里，而且要准确传输信道上的消息，不被窃听者窃听和篡改。这里的消息（Message）被称为明文（Plain Text），用某种方法伪装消息以隐藏它的内容的过程称为加密；加密后的消息称为密文（Cipher Text），而把密文转变为明文的过程称为解密。使消息保密的技术称为密码编码学，破译密文的技术和科学称为密码分析学。因此，密码学包括密码编码学和密码分析学两部分。

数据加密技术主要分为数据传输加密和数据存储加密。数据传输加密技术主要是对传输中的数据进行加密，常用的有链路加密、节点加密和端到端加密 3 种方式。

1）链路加密是传输数据仅在物理层上的数据链路层进行加密，不考虑信源和信宿，它用于保护通信节点间的数据。接收方是传送路径上的各台节点机，数据在每个节点机内都要被解密和再加密，依次进行，直至到达目的地。

2）与链路加密类似的节点加密方式是在节点处采用一个与节点机相连的密码装置，密文在该装置中被解密并被重新加密，明文不通过节点机，避免了链路加密节点处易受攻击的缺点。

3）端到端加密为数据从一端到另一端提供的加密方式。数据在发送端被加密，在接收端解密，中间节点处不以明文的形式出现。端到端加密是在应用层完成的。在端到端加密中，数据传输单位中除报头外的报文均以密文的形式贯穿于全部传输过程，只是在发送端和接收端才有加/解密设备，而在中间任何节点处报文均不解密。因此，不需要有密码设备，同链路加密相比，可减少密码设备的数量。另一方面，数据传输单位由报头和报文组成，报文为要传送的数据集合，报头为路由选择信息等（因为端到端传输中涉及路由选择）。在链路加密时，报文和报头两

者均须加密。而在端到端加密时，由于通路上的每一个中间节点虽不对报文解密，但为将报文传送到目的地，必须检查路由选择信息。因此，只能加密报文，而不能对报头加密。这样很容易被某些通信分析者发觉，而从中获取某些敏感信息。链路加密对用户来说比较容易，使用的密钥较少，而端到端加密比较灵活，对用户可见。在对链路加密中各节点安全状况不放心的情况也可使用端到端加密方式。

2．数据加密算法

明文用 M 或 P 表示，密文用 C 表示。密码算法又称密码，是用于加密和解密的函数。如果算法的保密性基于保持算法的秘密，这种算法称为受限的算法，受限制的算法不可能进行质量控制和标准化。现代密码学用密钥解决了这个问题，密钥用 K 表示，密钥 K 的取值范围称为密钥空间。基于密钥的算法分为对称算法（又称传统密码算法）和非对称算法（又称公开密钥算法）。

对称算法就是加密密钥能够从解密密钥中推算出来，反过来也成立。在多数对称算法中，加/解密密钥是相同的。这些算法又称秘密密钥算法或单密钥算法，它要求发送者和接收者在安全通信之前共同协商一个密钥。对称算法的安全性依赖于密钥的安全性，泄露密钥就意味着任何人都能对消息进行加/解密。只要通信需要保密，密钥就必须保密。

对称算法可以用图 5-3 表示，其中加密密钥与解密密钥相同。

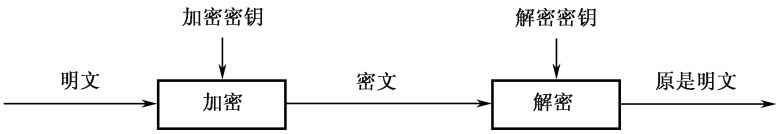


图 5-3　对称加密体制

下面重点介绍西泽密码。

西泽大帝是古罗马共和国末期著名的统帅和政治家。虽然他一生从未登上过皇位，但是直到今天在西方国家，他的名字是君主的代名词。他博学多才、文武双全，既是卓越的军事家，又是雄辩的文学家。密码最早就是应用在军事上的。在西泽大帝出征时，为了避免军令落入敌军手中而泄露秘密，他自己发明了一种单字替代密码。所谓“单字替代密码”就是把明文中的每个字母用密文中的对应字母替代，明文字符集与密文字符集是一一对应的关系。例如：

明文：ABCDEFGHIJKLMNOPQRSTUVWXYZ

密文：defghijklmnopqrstuvwxyzabc

如果指定一个数字元给每个字母（a=0，b=1，…，z=25），则可得明文 *P* 与密文 *C* 的关系式：

C=*E*（p）=（p+3）mod26，*P*=*D*（c）=（c−3）mod26

例如：明文为 CRACK IT，可得密文为 FUDFN LW。

我们可以移动任何长度，即 *C*=*E*（p）=（p+*K*）mod26，*K* 的范围是 1～25。

在西泽时代只有贵族才识字，要瞒天过海是很容易的。但是，在今天西泽密码变得很不安全。因为 *K* 仅有 25 种可能，只要知道是用西泽码加密，那尝试 25 次就可以得到明文。因此，

由西泽码又出现了其他的替代方法。在后面将介绍其他算法。

3．资料加密技术的发展

（1）密码专用芯片集成

密码技术是信息安全的核心技术，无处不在，目前已经渗透到大部分安全产品之中，正向芯片化方向发展。在芯片设计制造方面，目前微电子工艺已经发展到很高水平，芯片设计的水平也很高。我国在密码专用芯片领域的研究起步落后于国外，近年来我国集成电路产业技术的创新和自我开发能力得到了加强，微电子工业得到了发展，从而推动了密码专用芯片的发展。加快密码专用芯片的研制将会推动我国信息安全系统的完善。

（2）量子加密技术的研究

量子技术在密码学上的应用分为两类：一类是利用量子计算机对传统密码体制的分析；另一类是利用单光子的测不准原理在光纤一级实现密钥管理和信息加密，即量子密码学。量子计算机相当于一种传统意义上的超大规模并行计算系统，利用量子计算机可以在几秒内分解 RSA129 的公钥。根据互联网的发展，全光纤网络将是今后网络连接的发展方向，利用量子技术可以实现传统的密码体制，在光纤一级完成密钥交换和信息加密，其安全性是建立在 Heisenberg 的测不准原理上的，如果攻击者企图接收并检测信息发送方的信息（偏振），则将造成量子状态的改变，这种改变对攻击者而言是不可恢复的，而对收发方则可很容易地检测出信息是否受到攻击。目前量子加密技术仍然处于研究阶段（在我国处于领先地位），其量子密钥分配 QKD 在光纤上的有效距离还达不到距离光纤通信的要求。

5.2.2　数据加密标准 DES 与 IDEA 算法

1．数据加密标准 DES 的思想

1973 年，美国国家标准局（NBS）在认识到建立数据保护标准既明显又急迫的情况下，开始征集联邦数据加密标准的方案。1975 年 3 月 17 日，NBS 公布了 IBM 公司提供的密码算法，以标准建议的形式在全国范围内征求意见。经过两年多的公开讨论之后，1977 年 7 月 15 日，NBS 宣布接受这个建议，作为联系信息处理标准 46 号数据加密标准（Data Encryption Standard，DES）正式颁布，供商业界和非国防性政府部门使用。

根据密钥类型不同将现代密钥技术分为两类：一类是对称加密（秘密钥匙加密）系统，另一类是公开密钥加密（非对称加密）系统。目前最著名的对称加密算法有有效数据加密标准（DES）和欧洲数据加密标准（IDEA）等。随后 DES 成为全世界使用最广泛的加密标准。

对称式密码是指收发双方使用相同密钥的密码，而且通信双方都必须获得这把钥匙并保持钥匙的秘密。传统的密码都属于对称式密码。非对称式密码是指收发双方使用不同密钥的密码，现代密码中的公共密钥密码就属于非对称式密码。

对称加密算法的主要优点是加密和解密速度快，加密强度高，且算法公开，但其最大的缺点是实现密钥的秘密分发困难，在有大量用户的情况下密钥管理复杂，而且无法完成身份认证等功能，不便于应用在网络开放的环境中。加密与解密的密钥和流程是完全相同

的，区别仅仅是加密与解密使用的子密钥序列的施加顺序刚好相反。DES 密码体制的安全性应用不依赖于算法的保密，其安全性仅以加密密钥的保密为基础。

但是，经过 20 多年的使用，已经发现 DES 有很多不足之处，对 DES 的破解方法也日趋有效。AES（高级加密标准）将会替代 DES 成为新一代加密标准。

非对称加密算法的优点是能适应网络的开放性要求，密钥管理简单，并且可方便地实现数字签名和身份认证等功能，是目前电子商务等技术的核心基础。其缺点是算法复杂，加密数据的速度和效率较低。因此在实际应用中，通常将对称加密算法和非对称加密算法结合使用，利用 DES 或者 IDEA 等对称加密算法来进行大容量数据的加密，而采用 RSA 等非对称加密算法来传递对称加密算法所使用的密钥，通过这种方法可以有效地提高加密的效率，并能简化对密钥的管理。

对称密码系统的安全性依赖于以下两个因素：第一，加密算法必须足够强，仅仅基于密文本身去解密信息在实践上是不可能的；第二，加密算法的安全性依赖于密钥的秘密性，而不是算法的秘密性，因此我们没有必要确保算法的秘密性，却需要保证密钥的秘密性。对称加密系统的算法实现速度极快，从 AES 候选算法的测试结果看，软件实现的速度都达到了每秒数兆或数十兆比特。对称密码系统的这些特点使其有着广泛的应用。因为算法不需要保密，所以制造商可以开发出低成本的芯片以实现数据加密。这些芯片有着广泛的应用，适合于大规模生产。

对称加密系统最大的问题是密钥的分发和管理非常复杂、代价高昂。例如，对于具有 n 个用户的网络，需要 n（n−1）/2 个密钥，在用户群不是很大的情况下，对称加密系统是有效的。但是对于大型网络，当用户群很大并分布很广时，密钥的分配和保存就成了大问题。对称加密算法的另一个缺点是不能实现数字签名。

通过定期在通信网络的源端和目的端同时改用新的 Key，便能更进一步提高数据的保密性，这正是现在金融交易网络的流行做法。

公开密钥加密系统采用的加密钥匙（公钥）和解密钥匙（私钥）是不同的。由于加密钥匙是公开的，密钥的分配和管理就很简单。例如，对于具有 n 个用户的网络，仅需要 2n 个密钥。

公开密钥加密系统还能够很容易地实现数字签名。因此，其最适合于电子商务应用需要。在实际应用中，公开密钥加密系统并没有完全取代对称密钥加密系统，在实际应用中可利用二者各自的优点，采用对称加密系统加密文件，采用公开密钥加密系统加密“加密檔”的密钥（会话密钥），这就是混合加密系统。它较好地解决了运算速度问题和密钥分配管理问题。因此，公开密钥体制通常被用来加密关键性的、核心的机密数据，而称密码体制通常被用来加密大量的资料。

DES 算法在 POS、ATM、磁卡及智慧卡（IC 卡）、加油站、高速公路收费站等领域被广泛应用，以此来实现关键数据的保密。例如，信用卡持卡人的 PIN 的加密传输，IC 卡与 POS 间的双向认证、金融交易数据包的 MAC 校验等，均用到 DES 算法。

DES 算法的入口参数有 3 个：Key、Data 和 Mode。其中 Key 为 8B 共 64 位，是 DES 算法的工作密钥；Data 也为 8B 64 位，是要被加密或解密的数据；Mode 为 DES 的工作方式，有两种，即加密或解密。

DES 算法是这样工作的：如果 Mode 为加密，则用 Key 去把数据 Data 进行加密，生成 Data

的密码形式（64 位）作为 DES 的输出结果；如果 Mode 为解密，则用 Key 把密码形式的数据 Data 解密，还原为 Data 的明码形式（64 位）作为 DES 的输出结果。在通信网络的两端，双方约定一致的 Key，在通信的源点用 Key 对核心数据进行 DES 加密，然后以密码形式在公共通信网（如电话网）中传输到通信网络的终点，数据到达目的地后，用同样的 Key 对密码数据进行解密，便再现了明码形式的核心数据。这样，便保证了核心数据（如 PIN、MAC 等）在公共通信网中传输的安全性和可靠性。

通过定期地对通信网络的源端和目的端同时改用新的 Key，便能更进一步提高数据的保密性，这正是现在金融交易网络的流行做法。

对于详细算法的学习可以参考其他书目或者上网搜索，这里不再赘述。

2．IDEA 算法

IDEA 是一个迭代分组密码，分组长度为 64 位，密钥长度为 128 位。IDEA 密码中使用了以下 3 中不同的运算。

1）逐位异或运算。

2）模 216 加运算。

3）模 216+1 乘运算，0 与 216 对应。

IDEA 算法由 8 轮迭代和随后的一个输出变换组成。它将 64 位的数据分成 4 个子块，每个 16 位，令这 4 个子块作为迭代第一轮的输出，全部共 8 轮迭代。每轮迭代都是 4 个子块彼此间及 16 位的子密钥进行异或，模 216 加运算，模 216+1 乘运算。除最后一轮外，把每轮迭代输出的 4 个子块的第 2 和第 3 子块互换。该算法所需要的“混淆”可通过连续使用 3 个“不兼容”的群运算于两个 16 位子块来获得，并且该算法所选择使用的 MA（乘加）结构可提供必要的“扩散”。

5.2.3　病毒防护公开密钥算法

公开密钥算法（Public-Key Algorithm，又称非对称算法）：作为加密的密钥不同于作为解密的密钥，而且解密密钥不能根据加密密钥计算出来（至少在合理假定的长时间内）。之所以称为公开密钥算法，是因为加密密钥能够公开，即陌生人可以用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。

加密密钥又称公开密钥（Public Key，简称公钥），解密密钥称为私人密钥（Private Key，简称私钥）。

注意，上面说到的用公钥加密、私钥解密是应用于通信领域中的信息加密。在共享软件加密算法中，我们常用的是用私钥加密、公钥解密，即公开密钥算法的另一用途——数字签名。关于公开密钥算法的安全性，我们引用一段话：“公开密钥算法的安全性都是基于复杂的数字难题。根据所给予的数字难题来分类，有以下 3 类系统目前被认为是安全和有效的，即大整数因子分解系统（具有代表性的有 RSA）、离散对数系统（具有代表性的有 DSA、ElGamal）和椭圆曲线离散对数系统（具有代表性的有 ECDSA）。”

常见的公开密钥算法主要有以下几种。

1）RSA：能用于信息加密和数字签名。

2）ElGamal：能用于信息加密和数字签名。

3）DSA：能用于数字签名。

4）ECDSA：能用于信息加密和数字签名。

公开密钥算法将成为共享软件加密算法的主流，因为它的安全性好（当然还是作者的使用）。以 RSA 为例，当 N 的位数大于 1024 后（强素数），现在认为分解困难。公开密钥最主要的特点就是加密和解密使用不同的密钥，每个用户保存着一对密钥：公开密钥 PK 和秘密密钥 SK。因此，这种体制又称为双钥或非对称密钥密码体制。

在这种体制中，PK 是公开信息，用作加密密钥，而 SK 需要由用户自己保密，用作解密密钥。加密算法 E 和解密算法 D 也都是公开的。虽然 SK 与 PK 是成对出现的，但却不能根据 PK 计算出 SK。公开密钥算法的特点如下。

1）用加密密钥 PK 对明文 X 加密后，再用解密密钥 SK 解密，即可恢复出明文，或写为：DSK（EPK（X））=X。

2）加密密钥不能用来解密，即 DPK（EPK（X））≠X。

3）在计算机上可以容易地产生成对的 PK 和 SK。

4）从已知的 PK 可能推导出 SK。

5）加密和解密的运算可以对调，即 EPK（DSK（X））=X。

在公开的密钥密码体制中，最有名的一种是 RSA 体制。它已被 ISO/TC 97 的数据加密技术分委员会 SC 20 推荐为公开密钥数据加密标准。

1．RSA 公开密钥密码算法

1976 年，Diffie 和 Hellman 为解决密钥管理问题，在他们的奠基性工作“密码学的新方向”一文中，提出一种密钥交换协议，允许在不安全的媒体上通过通信双方交换信息，安全地传送秘密密钥。在此新思想的基础上，很快出现了非对称密钥密码体制，即公钥密码体制。在公钥密码体制中，加密密钥不同于解密密钥，加密密钥公之于众，谁都可以使用；解密密钥只有解密人自己知道。它们分别称为公开密钥和秘密密钥。在迄今为止的所有公钥密码体制中，RSA 系统是最著名、使用最多的一种。RSA 公开密钥密码系统是由 Rivest、Shamir 和 Adleman 于 1977 年提出的。RSA 的取名就是来自于这 3 位发明者的姓的第一个字母。

下面主要讲述 RSA 算法的基本要素。

RSA 的安全性依赖于大数分解。公开密钥和私有密钥都是两个大素数（大于 100 个十进制位）的函数。下面描述密钥对是如何产生的。

1）选择两个大素数 p 和 q，计算 $n=p\times q$，$\varphi(n)=(p-1)\times(q-1)$。

2）选择随机数和 $\varphi(n)$ 互质的数 d，要求 $d<\varphi(n)$。

3）利用 Euclid 算法计算 e，满足 $e\times d=1\bmod(p-1)\times(q-1)$，即 d 和 e 的乘积和 1 模 $\varphi(n)$ 同余。

4）于是，数（n，e）是加密密钥，（n，d）是解密密钥。两个素数 p 和 q 不再需要，应该丢弃，不要让任何人知道。

加密信息 m 时，首先把 m 分成等长数据块 m_1，m_2，…，m_i，块长 s，其中 $2^s \leqslant n$，s 尽可能大。对应的密文：

$$ci = (m_i)^e \bmod(n) \tag{5-1}$$

解密时做如下计算：

$$mi = (ci)^d \bmod(n) \tag{5-2}$$

RSA 也可用于数字签名，方案是用式（5-1）签名，用式（5-2）验证。具体操作时考虑到安全性和 m 信息量较大等因素，一般是先做 HASH 运算。

对于巨大的质数 p 和 q，计算乘积 $n = p \times q$ 非常简便，而逆运算却非常难，这是一种“单向性”，相应的函数称为“单向函数”。任何单向函数都可以作为某一种公开密钥密码系统的基础，而单向函数的安全性也就是这种公开密钥密码系统的安全性。

RSA 算法安全性的理论基础是大数的因子分解问题至今没有很好的算法，因而公开 e 和 n 不易求出 p、q 及 d。RSA 算法要求 p 和 q 是两个足够大的素数（如 100 位十进制数）且长度相差比较小。

为了说明该算法的工作过程，下面给出一个简单例子，显然在这里只能取很小的数字，但是如上所述，为了保证安全，在实际应用时，我们所用的数字要大得多。

【例 5-1】选取 p=3，q=11，则 n=33，$\varphi(n) = (p-1) \times (q-1) = 20$。选取 d=13[大于 p 和 q 的数，且小于 $\varphi(n)$，并与 $\varphi(n)$ 互质，即最大公约数是 1]，通过 $e \times 13 = 1 \bmod 20$，计算出 e=17[大于 p 和 q，与 $\varphi(n)$ 互质]。

假定明文为整数 M=8，则密文 C 为

$$C = M^e \bmod n = 817 \bmod 33 = 2251799813685248 \bmod 33 = 2$$

恢复原明文 M 为

$$M = C^d \bmod n = 213 \bmod 33 = 8192 \bmod 33 = 8$$

因为 e 和 d 互逆，公开密钥加密方法也允许采用这样的方式对加密信息进行“签名”，以便接收方能确定签名不是伪造的。

假设 A 和 B 希望通过公开密钥加密方法进行数据传输，A 和 B 分别公开加密算法和相应的密钥，但不公开密钥算法和相应的密钥。A 和 B 的加密算法分别是 ECA 和 ECB，解密算法分别是 DCA 和 DCB，ECA 和 DCA 互逆，ECB 和 DCB 互逆。若 A 要向 B 发送明文 P，不是简单地发送 ECB（P），而是先对 P 施以其解密算法 DCA，再用加密算法 ECB 对结果加密后发送出去。

密文 C 为 C=ECB（DCA（P））。

B 收到 C 后，先后施以其解密算法 DCB 和加密算法 ECA，得到明文 P。

```
ECA（DCB）=ECA（DCB（ECB（DCA（P））））    //DCB 和 ECB 相互抵消//
          =ECA（DCA（P））                  //DCA 和 ECA 相互抵消//
          =P
```

这样 B 就确定报文确实是从 A 发出的，因为只有当加密过程利用了 DCA 算法，用 ECA 才能获得 P，只有 A 才知道 DCA 算法，任何人即使是 B 也不能伪造 A 的签名。

2．RSA 的实用性

在 CA 系统中，公开密钥系统主要用于对秘密密钥（在 CA 系统中称控制字，用于对数据码流进行加扰）的加密过程。每个用户如果想要对数据进行加密和解密，都需要生成一对自己的密钥对（Key Pair）。密钥对中的公开密钥和非对称加密解密算法是公开的，但私有密钥则应该由密钥的主人妥善保管。对数据信息进行加密传输的实际过程是：发送方生成一个秘密密钥并对数据流用秘密密钥（控制字）进行加扰，然后用网络把加扰后的数据流传输到接收方。

发送方生成一对密钥，用公开密钥对秘密密钥（控制字）进行加密，然后通过网络传输到接收方。

接收方用自己的私有密钥（存放在接收机智能卡中）进行解密后得到秘密密钥（控制字），然后用秘密密钥（控制字）对数据流进行解扰，得到数据流的解密形式。

因为只有接收方才拥有自己的私有密钥，所以其他人即使得到了经过加密的秘密密钥（控制字），也因为无法进行解扰而保证了秘密密钥（控制字）的安全性，从而也保证了传输数据流的安全性。实际上，在数据传输过程中实现了两个加密解密过程，即数据流本身的加解扰和秘密密钥（控制字）的加密解密，这分别通过秘密密钥（控制字）和公开密钥来实现。

3．RSA 的实用考虑

RSA 公开密钥密码体制的安全性取决于从公开密钥（n，e）计算出私有密钥（n，d）的困难程度，而后者则等同于从 n 找出它的两个质因数 p 和 q。因此，寻求有效的因数分解的算法就是寻求击破 RSA 公开密钥密码系统的关键。

显然，选取大数 n 是保障 RSA 算法的一种有效办法。RSA 实验室认为，512 位的 n 已不够安全，1997 年或 1998 年后应停止使用。他们建议，现在的个人应用需要用 768 位的 n，公司要用 1024 位的 n，极其重要的场合应该用 2048 位的 n。

不对称密钥密码体制（即公开密钥密码体制）与对称密钥密码体制相比较，确实有其不可取代的优点，但它的运算量远大于后者，超过几百倍、几千倍，要复杂得多。

在公共媒体网络上全部用公开密钥密码体制来传送机密信息是没有必要的，也是不现实的。在计算机系统中使用对称密钥密码体制已有多年，既有比较简便可靠的、久经考验的方法，如以 DES（数据加密标准）为代表的分块加密算法（及其扩充 DESX 和 Triple DES）；又有一些新的方法发表，如由 RSA 公司的 Rivest 研制的专有算法 RC2、RC4 和 RC5 等，其中 RC2 和 RC5 是分块加密算法，RC4 是数据流加密算法。

如果传送机密信息的网络用户双方使用某个对称密钥密码体制（如 DES），同时使用 RSA 不对称密钥体制来传送 DES 的密钥，就可以综合发挥两种体制的优点，即 DES 的高速简便性和 RSA 密钥管理的方便和安全性。

另外，RSA 算法还有以下缺点。

1）产生密钥很麻烦。共受到素数产生技术的限制，因而难以做到一次一密。

2）安全性差。RSA 的安全性依赖于大数的因子分解，但并没有从理论上证明破译 RSA 的难度与大数分解难度等价，而且密码学界多数人士倾向于因子分解而不是 NPC 问题。目前，人们已能分解 140 多个十进制位的大素数，这就要求使用更长的密钥，速度更慢；另外，目前人

们正在积极寻找攻击 RSA 的方法，如选择密文攻击，一般攻击者是将某一信息做伪装（Blind），让拥有私钥的实体签署。然后，经过计算就可得到它所想要的信息。实际上，攻击利用的都是同一个弱点，即存在这样一个事实：乘幂保留了输入的乘法结构。

$$(XM)^d = X^d \times M^d \bmod n$$

前面已经提到，这个固有的问题来自于公钥密码系统的最有用特征——每个人都能使用公钥。但从算法上无法解决这一问题，主要措施有两条：一条是采用好的公钥协议，保证工作过程中实体不对其他实体任意产生的信息解密，不对自己一无所知的信息签名；另一条是绝不对陌生人送来的随机文档签名，签名时首先使用 One-Way Hash Function 对文档做 HASH 处理，或同时使用不同的签名算法。除了利用公共函数，人们还尝试利用解密指数或 $\varphi(n)$ 等攻击。

3）速度太慢。由于 RSA 的分组长度太长，为了保证安全性，n 至少也要 600 位以上，运算代价很高，尤其是速度较慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化。目前，SET（Secure Electronic Transaction）协议中要求 CA 采用 2048 位的密钥，其他实体使用 1024 位的密钥。为了提高速度，目前人们广泛使用单、公钥密码结合使用的方法，使优缺点互补：单钥密码加密速度快，人们用它来加密较长的文件，然后用 RSA 来给文件密钥加密，极好地解决了单钥密码的密钥分发问题。

5.2.4　密钥管理和交换技术

1．密钥管理问题

密钥既然要求保密，这就涉及密钥的管理问题。任何保密都是相对的，而且是有时效的。要管理好密钥还要注意以下几个方面。

（1）使用密钥时要注意时效和次数

如果用户可以一次又一次地使用同样的密钥与别人交换信息，那么密钥也同其他任何密码一样存在着一定的安全性问题，虽然说用户的私钥是不对外公开的，但是也很难保证长期不泄露。使用一个特定密钥加密的信息越多，提供给窃听者的材料也就越多，从某种意义上来讲也就越不安全。因此，一般强调仅将一个对话密钥用于一条信息或一次对话中，或者建立一种按时更换密钥的机制以减小密钥泄露的可能性。

（2）多密钥管理

在大企业中，要使任意两人之间进行秘密对话，就需要每个人记住很多密钥。Kerberos 提供了一种较好的解决方案，它使密钥的管理和分发变得十分容易，虽然这种方法本身还存在一定的缺点，但它建立了一个安全的、可信任的密钥分发中心，每个用户只要知道一个和 KDC 进行会话的密钥即可。

假设用户甲想要和用户乙进行秘密通信，则用户甲先和 KDC 通信，用只有用户甲和 KDC 知道的密钥进行加密，用户甲告诉 KDC 他想和用户乙进行通信，KDC 会为用户甲和用户乙之间的会话随机选择一个对话密钥，并生成一个标签，这个标签由 KDC 和用户乙之间的密钥进行加密，并在用户甲和用户乙对话时，由用户甲把这个标签交给用户乙。这个标签的作用是让用户甲确信和他交谈的是用户乙，而不是冒充者。因为这个标签是由只有用户乙和 KDC 知道的密

钥进行加密的，所以即使冒充者得到用户甲发出的标签也不可能进行解密，只有用户乙收到后才能够进行解密，从而确定了与用户甲对话的人就是用户乙。

当 KDC 生成标签和随机会话密码时，就会把它们用只有用户甲和 KDC 知道的密钥进行加密，然后把标签和会话密钥传给用户甲，加密的结果可以确保只有用户甲能得到这个信息，只有用户甲能利用这个会话密钥和用户乙进行通话。同理，KDC 会把会话密码用只有 KDC 和用户乙知道的密钥加密，并把会话密钥给用户乙。

用户甲启动一个和用户乙的会话，并用得到的会话密钥加密自己和用户乙的会话，还要把 KDC 传给它的标签传送给用户乙以确定用户乙的身份，然后用户甲和用户乙之间即可用会话密钥进行安全会话，而且为了保证安全，这个会话密钥是一次性的，这样黑客就更难进行破解。同时由于密钥是一次性由系统自动产生的，因此用户不必记很多密钥而方便了人们的通信。

2．Diffie-Hellman 密钥交换技术

密钥交换指明如何在客户端和服务器的数据传输中使用共享的对称密钥。

DH 算法是 Diffie 和 Hellman 提出的。此算法是最早的公钥算法。它实质是一个通信双方进行密钥协定的协议：两个实体中的任何一个使用自己的私钥和另一实体的公钥，得到一个对称密钥，其他实体都计算不出来这一对称密钥。DH 算法的安全性基于有限域上计算离散对数的困难性。离散对数的研究现状表明：所使用的 DH 密钥至少需要 1024 位，才能保证有足够的中、长期安全。

DH 算法是用于密钥交换的最早、最安全的算法之一。DH 算法的基本工作原理：通信双方公开或半公开交换一些准备用来生成密钥的“材料数据”，在彼此交换过密钥生成“材料”后，两端可以各自生成出完全一样的共享密钥。在任何时候，双方都绝不交换真正的密钥。

DH 密钥交换协议如下。

首先，Alice 和 Bob 双方约定两个大整数 n 和 g，其中 $1<g<n$，这两个整数无须保密，然后执行以下过程。

1）Alice 随机选择一个大整数 x（保密），并计算 $X = g^x \bmod n$ 。

2）Bob 随机选择一大整数 x（保密），并计算 $Y = g^y \bmod n$ 。

3）Alice 把 X 发送给 Bob，Bob 把 Y 发送给 Alice。

4）Alice 计算 $K = Y^x \bmod n$ 。

5）Bob 计算 $K = X^y \bmod n$ 。

K 即是共享的密钥。

监听者 Eve 上只能监听到 X 和 Y，但无法通过 X 和 Y 计算出 x 和 y，因此，Eve 无法计算出 $K = g^{xy} \bmod n$ 。

3．RSA 密钥交换技术

RSA 适用于数字签名和密钥交换。RSA 加密算法是目前应用最广泛的公钥加密算法，特别适用于通过 Internet 传送的数据。RSA 算法的安全性基于分解大数字时的困难（就计算机处理能力和处理时间而言）。在常用的公钥算法中，RSA 与众不同，它能够进行数字签名和密钥交换运算。

DSA 仅适用于数字签名。数字签名算法（Digital Signature Algorithm，DSA）由美国国家安全署（United States National Security Agency，NSA）发明，已经由美国国家标准与技术协会（National Institute of Standards and Technology，NIST）收录到联邦信息处理标准（Federal Information Processing Standard，FIPS）之中，作为数字签名的标准。DSA 算法的安全性源自计算离散算法的困难。这种算法仅用于数字签名运算（不适用于数字加密）。

DH 仅适用于密钥交换。DH 算法的安全性源自在一个有限字段中计算离散算法的困难。DH 算法仅用于密钥交换。

5.2.5　密码分析与攻击

密码分析是在不知道密钥的情况下恢复出明文的科学。密码分析也可以发现密码体制的弱点。

传统的密码分析技术主要基于穷尽搜索。它破译 DES 需要若干人/年的时间。

现在密码分析技术包括差分密码分析技术、线性密码分析技术和与密钥相关的密码分析。它改善了破译速度，但是破译速度还是很慢。

新一代密码分析技术主要基于物理特征的分析技术，它们包括电压分析技术、故障分析技术、侵入分析技术、时间分析技术、简单的电流分析技术、差分电流分析技术、电磁辐射分析技术、高阶差分分析技术和汉明差分分析技术。利用这些技术，攻击者可以在获得密码算法运行载体（计算机、保密机、加密盒、IC 卡等）的情况下，快速地获得密钥，从而破译整个密码系统。例如，破译 IC 卡的 DES 只需 10min。

这些技术在 1998 年被验证简单、实用之后，世界上的密码学家、半导体厂商、IC 卡生产厂商、政府及军事机构都集中人力和物力研究、查验抵抗技术。人们已经发明了许多专利和技术来抵抗这些分析技术，但是，这些专利和技术只增加了这些分析的困难性，并没有实际的技术能力来完全抵抗这些攻击。

例如，使用随机数的方法只是增加了这些分析的难度，但却严重地增加了加密和解密的时间，并且这些分析技术同样可以破译这种使用随机数的方法。

物理上使用硬件做加密线路并没有根本上抵抗这种攻击。西门子公司在存储器中采用加密方式，并且使用硬件加密总线。然而，真正在加密和解密中所用的密钥仍然以明文的形式在 CPU 中出现，分析者可以用 DPA 方法获得该密钥。

基于物理特征的密码分析技术从 1996 年开始有公开发表的学术论文，第一批研究成果是韩永飞等人和 Shamir 等人分别独立发表的基于存储体出错的密码分析方法。1996 年底，Cambridge 大学的 Anderson 等人发表了基于不同电压的密码分析方法，美国人 Butch 发表的基于时间和基于差分电压分析的密码分析技术震动了信息安全领域。从 1999 年起，发达国家的政府开始组织力量研究基于物理特征的密码分析技术。

1．基于密文的攻击

（1）唯密文攻击

密码分析者有一些消息密文，这些消息都用同一加密算法加密。密码分析者的任务是恢复

尽可能多的明文，或者最好能推算出加密消息的密钥，以便采用相同的密钥解算出其他被加密的消息。

（2）已知密文攻击

密码分析者不仅可以得到一些消息的密文，而且也知道这些消息的明文。分析者的任务就是用加密信息推出用来加密的密钥或导出一个算法，此算法可以对用同一密钥加密的任何新的消息进行解密。

（3）选择密文攻击

密码分析者能选择不同的被加密的密文，并可得到对应的解密明文。例如，密码分析者选择一个防篡改的自动解密盒，其任务是推出密钥。这种攻击主要用于公开密钥算法，选择密文攻击有时也可有效用于对称算法（有时选择明文攻击和选择密文攻击一起称为选择文本攻击）。

这个攻击的前提是分析者能够获得一个密封的解密盒，也就是一个已经固化的专门用于对用某一个特定密钥加密过的密文进行解密的硬件。攻击的方法就是随机产生一个伪密文（不一定是合法的），让解密盒进行解密，将所得到的明文和密文进行比较，得到关于密钥或者算法的相关信息。这种攻击实际上和选择明文攻击类似（就是它的逆过程），只是明文变成密文后泄露密钥信息是比较主要的，但是从明文里面泄露消息则考虑得相对较少。此外，如果是非对称加密算法，两个破解方向由于密钥长度的不同会引起破解难度的巨大差别。因此选择密文攻击很可能得到比选择明文攻击更多的信息。

2．基于明文的密码攻击

（1）选择明文攻击

分析者不仅可得到一些消息的密文和相应的明文，而且他们也可选择加密的明文。这比一个明文攻击更有效。因为密码分析者能选择特定的明文块去加密，这些块可能产生更多关于密钥的信息，分析者的任务是推出用来加密消息的密钥或者导出一个算法，此算法可以对用同一密钥加密的任何新的消息进行解密。

（2）自适应选择明文攻击

这是选择明文攻击的特殊情况。密码分析者不仅能选择被加密的密文，而且也能给以前加密的结果修正这个选择。选取较小的明文块，然后再基于第一块的结果选择另一块明文块，以此类推。

3．中间人攻击

中间人攻击（Man-in-the-Middle Attack）是一种“间接”的入侵攻击，这种攻击模式是通过各种手段将受入侵控制的一台计算机虚拟放置在网络连接中的两台通信计算机之间，这台计算机就称为“中间人”。然后入侵者把这台计算机模拟为一台或两台原始计算机，使“中间人”能够与原始计算机建立活动连接并允许其读取或修改传递的信息，然而两台原始计算机用户却认为它们是在互相通信。通常，这种“拦截数据—修改数据—发送数据”的过程就被称为“会话劫持”（Session Hijack）。

中间人攻击是黑客的惯用伎俩。此技术将一台计算机放置在网络连接中的两台通信计算机之间，然后用中间计算机模拟一台或两台原始计算机。此技术可以使“中间人”与原始计算机建立

活动连接并允许其读取和/或修改传递的信息，然而两台原始计算机用户却认为它们在互相通信。

一些 Internet 服务提供商（ISP）开发了试图对抗中间人攻击和电子邮件欺骗的过滤做法。例如，许多 ISP 只授权用户通过 ISP 服务器发送电子邮件，并且根据对抗垃圾邮件的需要来验证此限制。但是，此项限制同样阻止了授权用户使用第三方提供的合法电子邮件服务，这使得许多高级用户十分不满。一些电缆 ISP 尝试阻止音频或视频通信，试图强制用户使用自己的 IP 语音或视频流服务。其他实例包括试图禁止某些形式的 VPN 通信，原因是 VPN 是一项需要较高订购费用的商务服务，并禁止用户在家中运行服务器。

ISP 筛选器通常通过使用路由器的硬件功能来实施，此类路由器在特定协议类型（用户数据协议或 TCP）、端口号或 TCP 标识（初始连接包，且无数据或确认码）上运行。如果使用 IPSec 有效地禁止了此类过滤，这就给 ISP 带来了两个非常极端的选择：要么禁止所有 IPSec 通信，要么禁止与某些已标识的对等端之间的通信。如果广泛使用了 IPSec，则这两种选择都将造成消费者的对抗性反应。

5.3　病毒防护

病毒是网络系统最直接的安全威胁，如何防范病毒的攻击和破坏是网络安全技术的重要方面。本节从病毒原理、病毒攻击和病毒发展多个视角介绍病毒防护技术。

5.3.1　病毒简介

1．病毒的定义

计算机病毒是一个程序，一段可执行码。像生物病毒一样，计算机病毒有其独特的复制能力，可以很快地蔓延，又常常难以根除。它们能把自身附着在各种类型的文件上，当文件被复制或从一个用户传送到另一个用户时，它们就随同文件一起蔓延起来。现在，随着计算机网络的发展，计算机病毒和计算机网络技术相结合，蔓延的速度更加迅速。

在生物学中，病毒是指侵入动植物体等有机生命体中，具有感染性、潜伏性、破坏性的微生物，而且不同的病毒具有不同的诱发因素。“计算机病毒”一词是人们联系到破坏计算机系统的“病原体”具有与生物病毒相似的特征，借用生物学病毒而使用的计算机术语。“计算机病毒”一词最早出现在美国作家 Thomas J. Ryan 于 1977 年出版的科幻小说 *The Adolescence of P-1* 中。

1983 年，美国计算机安全专家 Frederick Cohen 博士首次提出计算机病毒的存在。他认为：计算机病毒是一个能感染其他程序的程序，它靠修改其他程序，并把自身的副本嵌入其他程序而实现病毒的感染。1989 年，他进一步将计算机病毒定义为：病毒程序通过修改其他程序的方法将自己的精确副本或可能演化的形式放入其他程序中，从而感染它们。所谓感染，是指病毒将自身嵌入到指令序列中，致使执行合法程序的操作招致病毒程序的共同执行，或是以病毒程序的执行取而代之。

然而，对于什么是计算机病毒，从不同角度出发，还会有很多不同的定义。目前，国内比较流行的是采用 1994 年颁布的《中华人民共和国计算机信息系统安全保护条例》第 28 条定义，即“计算机病毒，是指编制或者在计算机程序中插入的破坏计算机功能或者毁坏数据，影响计算机使用，并能够自我复制的一组计算机指令或者程序代码”。

2．病毒的特征

计算机病毒是一段特殊的程序，它与生物学病毒有着十分相似的特征。除了与其他程序一样，可以存储和运行外，计算机病毒还有传染性、传播性、破坏性、寄生性、欺骗性、隐蔽性、潜伏性和衍生性等特征。它一般都隐藏在合法程序（被感染的合法程序称为宿主程序）中，当计算机运行时，它与合法的程序争夺系统的控制权，从而对计算机系统实施干扰和破坏作用。

一般病毒具有以下特征。

（1）可执行性

计算机病毒是一段能够执行的代码，既可以是二进制代码（如正常使用的带有图标的程序），也可以是一段脚本（如有些网页病毒是利用网页的脚本进行破坏，或者宏病毒利用 Office 对 Word 文档进行操作的时候获得执行权限，执行脚本进行破坏）。

（2）传染性与传播性

最早的计算机病毒或者狭义的计算机病毒的定义，都是强调恶意程序的传染性或者传播性。病毒要进行传染，就需要进行自我复制，常用的方法就是把自己的病原体代码注入到宿主程序中，当宿主程序运行时，病毒代码也能随之运行。病毒的传染性主要作用于一台主机上，而病毒要从一台主机蔓延到另一台主机，其所需要的另一种特征就是传播性，即通过网络或者存储介质（包括软盘、硬盘、光盘）进行传播。从目前的情况来看，通过网络进行传播的病毒已经占了绝大多数，由于网络的飞速发展和网络传播信息的快速性，导致快速传播成为病毒目前的重要特征。

（3）破坏性

计算机病毒的破坏性是多种多样的，如损坏数据、导致系统异常、窃取用户数据、阻塞网络等。总的来说，其破坏行为就是程序做了用户预料之外的事件。

（4）寄生性

狭义的病毒一般不是完整的程序，它通过附加在其他程序中，就像生物界中的寄生现象。被寄生的程序被称为宿主程序，或者称为病毒载体。当然，现在的某些病毒本身就是一段完整的程序，特别是广义病毒中的网络蠕虫。

（5）欺骗性

计算机病毒需要在受害者的计算机上获取可执行的权限，因为病毒首先要能执行，才能进行传染或者破坏。要获得可执行的权限，必须通过用户运行，或者通过系统直接运行。黑客常常会把带有病毒程序的名字命名为一些用户比较关心的程序名字，如把自己命名为微软补丁或者动画的名字，欺骗用户执行这个程序。“爱虫”病毒就是利用标题为“I LOVE YOU”的邮件欺骗用户。

（6）隐蔽性和潜伏性

计算机病毒要获得有效的传染和传播，就应该尽量在用户能够察觉的范围之外进行。因为用户一旦发现，计算机病毒就有可能被清除，这将影响到计算机病毒本身的生存性。因此，大多数病毒都把自己隐藏起来。隐藏起来的方式有很多，如把自己复制到 Microsoft Windows 目录下，或者复制到一般用户不会打开的目录下，如回收站、系统的临时目录，然后把自己的名字改成系统的文件名，或者与系统文件名类似。那么，用户运行的时候，就不会发现这个文件是一个病毒文件，而会认为这是一个系统文件，达到隐蔽的目的。另外，计算机病毒在运行后，一般以服务、后台程序、注入线程或钩子驱动程序的形式存在，驻留内存，不易被发现。潜伏性是指病毒在相当长的一段时间里，虽然在系统中存在，但不执行它的破坏功能，使用户难以觉察，而只有到达某个时间点或受其他条件的激发时才执行恶意代码。

（7）衍生性

既然计算机病毒是一段特殊的程序，了解病毒程序的人就可以根据其个人意图随意改动，从而衍生出另一种不同于原版病毒的新病毒。这种衍生出的病毒可能与原先的计算机病毒有很相似的特征，所以被称为原病毒的一种变种；如果衍生的计算机病毒已经与以前的计算机病毒有了很大（甚至是根本性）的差别，则此时就会将其认为是一种新的计算机病毒。变种或新的计算机病毒可能比原计算机病毒有更大的危害性。

3．病毒的防治

病毒严重干扰了人们对计算机的使用，各种病毒防治的方法也不断涌现。一般常用的防治方法有以下几种。

（1）把各种查杀病毒的新技术应用于反病毒软件

传统的病毒防治软件绝大多数采用的是特征码扫描技术，而加密变形病毒的出现使特征码扫描技术一筹莫展。反病毒的新技术（如启发式检测、虚拟机技术、实时监控技术、人工智能陷阱等）不断融入病毒防治软件。

（2）网络杀毒

现在病毒传播的主要手段通过网络进行，在本地网络的入口设置病毒防治系统，可以有效防止病毒进入本地局域网。一般可以在路由器、防火墙中加入反病毒模块，专门针对网络蠕虫、邮件病毒和网页恶意代码。

（3）个人防火墙

新型病毒的出现使人们逐渐认识到，网络安全已经不再是单纯的病毒问题，现在黑客的泛滥已经成为网络安全的另一大问题。因此，许多消费者在购买杀毒软件以后，又买了一套防火墙软件。所带来的问题是两款软件能否兼容。一方面，反病毒厂商需积极地和防火墙厂商开展各种合作；另一方面，反病毒厂商开发出自己的个人防火墙，在产品上实现反病毒和防黑客的二合一，使用户的信息能够得到双重的安全保障。目前，很多个人防火墙都具有病毒防治功能。

（4）邮件杀毒

电子邮件在带来方便和快捷的同时，也充当了病毒传播的主要工具。那些极具破坏力的病毒，如“梅丽莎”、“爱虫”、“红色代码Ⅱ”、“尼姆达”等，都把邮件作为一个重要的传播渠道。对于一款杀毒软件而言，“邮件病毒”的功能必不可少。病毒防治的一个重要方法就是堵住灾害

的源头，在病毒进入系统之前发现并杀除它。邮件杀毒可分为对已经收到的邮件进行查杀和接收邮件时的实时查杀。

（5）数据备份拯救系统

能杀灭所有病毒的软件不一定就是最好的软件。因为对任何人而言，计算机中的信息才是最为重要，防毒软件所应做的，除了要将病毒悉数杀灭，更重要的是要将遭到病毒破坏的数据信息恢复到正常状态。因此，“数据备份拯救系统”应该成为病毒防治软件必备的功能部件。

5.3.2 病毒原理

1．病毒的分类

按照计算机病毒的诸多特点及特性，其分类方法有很多种，所以同一种病毒按照不同的分类方法可能被分到许多不同的类别中。

（1）按攻击的操作系统分类

1）攻击 DOS 系统的病毒。这种病毒又称 DOS 病毒，出现最早，变种也最多，传播也非常广泛，如小球病毒等。

2）攻击 Windows 系统的病毒。这种病毒又称 Windows 病毒，随着 Windows 系统取代 DOS 系统成为 PC 的主流平台，Windows 系统也成为病毒攻击的主要对象。攻击 Windows 的病毒主要是宏病毒，有感染 Word 的宏病毒，有感染 Excel 的宏病毒，还有感染 Access 的宏病毒，其中感染 Word 的宏病毒最多。

3）攻击 UNIX 或 OS/2 系统的病毒。目前，UNIX 操作系统的应用非常广泛，许多大型的操作系统均采用 Unix 作为其主要的操作系统，所以攻击 Unix 大家族的病毒对人类的信息处理是一个严重的威胁。世界上也已经发现攻击 OS/2 系统的病毒。

（2）按传播媒介分类

1）单机病毒。单机病毒的载体是磁盘，常见的是病毒从软盘传入硬盘，感染系统，然后再感染其他软盘，进而再感染其他系统，早期的病毒都属于此类。

2）网络病毒。网络病毒的传播媒介是网络。当前，Internet 在世界上发展迅速，上网已成为计算机使用者的时尚。随着网上用户的增加，网络病毒的传播速度更快，范围更广，造成的危害更大。网络病毒往往造成网络堵塞，修改网页，甚至与其他病毒结合修改或破坏文件。例如，GPI 病毒是世界上第一个专门攻击计算机网络的病毒，最近的 CIH、Sircam、Code Red、Code Red Ⅱ、Code Blue、Nimda.a 等病毒在计算机网络上肆虐的程度越来越严重，已经成为病毒中危害最为严重的种类。如今的病毒大多都是网络病毒。

（3）按链接方式分类

计算机病毒需要进入系统，从而进行感染和破坏，因此，病毒必须与计算机系统内可能被执行的文件建立连接。这些被连接的文件可能是操作系统文件，可能是以各种程序设计语言编写的应用程序，也可能是应用程序所用到的数据文件（如 Word 文档）。根据病毒对这些文件的链接形式不同来划分病毒，可以分为如下几类。

1）源码型病毒。这类病毒在高级语言（如 Fortran、C、Pascal 等语言）编写的程序被编译

之前，插入目标源程序之中，经编译，成为合法程序的一部分。这类病毒程序一般寄生在编译处理程序或链接程序中。目前，这种病毒并不多见。

2）入侵型病毒。入侵型病毒又称嵌入型病毒，在感染时往往对宿主程序进行一定的修改，通常是寻找宿主程序的空隙将自己嵌入进去，并变为合法程序的一部分，使病毒程序与目标程序成为一体。这类病毒编写起来很难，要求病毒能自动在感染目标程序中寻找恰当的位置，把自身插入，同时还要保证病毒能正常实施攻击，且感染的目标程序能正常运行。一旦病毒侵入宿主程序，对其杀毒是十分困难的，清除这类病毒时往往会破坏合法程序。这类病毒的数量不多，但破坏力极大，而且很难检测，有时即使查出病毒并将其杀除，但被感染的程序也已经被破坏，无法再使用。

3）外壳型病毒。这类病毒程序一般链接在宿主程序的首尾，对原来的主程序不做修改或仅做简单修改。当宿主程序执行时，首先执行并激活病毒程序，使病毒得以感染、繁衍和发作。这类病毒易于编写，数量也最多。

4）操作系统型病毒。这类病毒程序用自己的逻辑部分取代一部分操作系统中的合法程序模块，从而寄生在计算机磁盘的操作系统区，在启动计算机时，能够先运行病毒程序，然后再运行启动程序，这类病毒可表现出很强的破坏力，可以使系统瘫痪，无法启动。

（4）按表现（破坏）情况分类

1）良性病毒。良性病毒是指那些只表现自己，而不破坏计算机系统的病毒。它们多出自一些恶作剧者之手。病毒制造者编写病毒的目的不是为了对计算机进行破坏，而是为了显示他们在计算机编写方面的技巧和才华。但这种病毒还是会干扰计算机系统的正常运行，占用计算机资源，而且违背计算机用户的意愿，所以也是应该被坚决制止的。再者，有些良性病毒也会由于交叉感染或编写方面的失误而造成不可估量的损失。

2）恶性病毒。恶意病毒的目的就是有意或无意地破坏系统中的信息资源。常见的恶意病毒的破坏行为是删除计算机系统内存储的数据和文件；也有一些恶意病毒不删除任何文件，而是对磁盘乱写一气，表面上看不出病毒破坏的痕迹，但文件和数据的内容已被改变；还有一些恶意病毒对整个磁盘或磁盘的特定扇区进行格式化，使磁盘的信息全部消失。而 CIH 病毒更加恶毒，它不仅能够破坏计算机系统内的数据，还能破坏计算机硬件，损坏某些机型的主板，这也是第一个被发现的可以破坏主板的病毒。

（5）按寄生方式分类

计算机病毒按其寄生方式大致可分为 3 类：引导型病毒、文件型病毒、混合型病毒。

1）引导型病毒。引导型病毒又称磁盘引导型、引导扇区型、磁盘启动型、系统型病毒等。引导型病毒就是把自己的病毒程序放在软磁盘的引导区及磁盘的主引导记录区或引导扇区，当作正常的引导程序，而将真正的引导程序搬到其他位置。这样，计算机启动时，就会把引导区的病毒程序当作正常的引导程序来运行，使寄生在磁盘引导区的静态病毒进入计算机系统，病毒变成活跃状态（或称病毒激活），这时病毒可以随时进行感染和破坏。此外，这种病毒体通常会改写硬盘上的主引导记录区、引导区、文件分配表、文件目录区、中断向量表等。

2）文件型病毒。文件型病毒是指所有通过操作系统的文件系统进行感染的病毒。文件型病毒以感染可执行文件（.bat、.exe、.com、.sys、.dll、.ovl、.vxd 等）的病毒为主，还有一些病毒

可以感染高级语言程序的源代码、开发库或编译过程中所生成的中间文件。病毒也可能隐藏在普通的数据文件中，但是这些隐藏在数据文件中的病毒不是独立存在的，必须要隐藏在可执行文件中的病毒部分来加载这些代码。宏病毒在某种意义上可以被看做文件型病毒，但由于其数量多、影响大，而且也有自己的特点，所以通常单独分类。

3）混合型病毒。混合型病毒又称综合型、复合型病毒，既具有引导型病毒的特点，又具有文件型病毒的特点，即这种病毒既可以感染磁盘引导扇区，又可以感染可执行文件。这类病毒的危害性更大。感染了混合型病毒的机器，如果只解除了文件上的病毒，而未解除硬盘上引导区的病毒，系统引导时又将病毒调入内存，会重新感染文件；如果只解除了主引导区的病毒，而可执行文件上的病毒没解除，只要执行带毒的文件，就会将硬盘上的主引导区感染。常常因为杀毒不彻底，而造成“病毒杀不死”的假象。

2．宏病毒

要分析宏病毒，首先要了解什么是“宏”。在 Microsoft 公司的手册中说明：如果需要在 Office 软件中反复进行某些工作，可以利用宏来自动完成这项工作。宏是一系列组合在一起的命令和指令，它们形成一个命令，以实现任务执行的自动化。用户可以创建并执行宏（宏实际上就是一条自定义的命令），以替代人工进行的一系列费时而单调的重复性操作，自动完成所需任务。

什么又是“宏病毒”呢？宏病毒是一种存储在文档、模板或加载宏程序中的计算机病毒。当打开已受感染的文件（如 Word 文档）或执行触发宏病毒的操作时，病毒就会被激活，并存储到 Normal.dot 模板或 Personal.xls 文件中。从此以后，保存的每一个文档都会自动被病毒“感染”，如果其他人打开受病毒感染的文件，那么宏病毒就会传播到他们的计算机中。

显然，宏病毒是只感染 Microsoft 的文档（.doc 或.xls 等）的一种专向病毒。宏病毒与攻击 DOS/Windows 可执行程序的病毒的机理完全不一样，它是用 VB 高级语言编写的病毒代码，直接混杂在文件中，并加以传播。宏病毒程序编写简单，对于现有的各种宏病毒，利用 Word 的 Visual Basic 编译器略做修改就会产生新的变种病毒。

要进一步了解宏病毒的活动，有必要先了解 Word 软件的工作工程。为了使 Word 更易于使用，Microsoft 公司在 Word 中集成了许多模块，如传真、报告、电子邮件、通讯录模板等。这些模板不仅包含了相应类型文档的一般格式，还允许用户在模板中添加宏，以使用户在制作自己的特定格式文件时，减少重复劳动。在所有这些模板中，最常见的就是 Normal.dot 模板（通用模板），它是启动 Word 时载入的默认模板。任何一个 Word 文件，其背后都有相应的模板，我们打开或建立大多数 Word 文档时，系统都会自动装入 Normal.dot 模板，并执行其中的宏。Word 打开文件时，首先要检查文件中包含的宏是否有自动执行的宏（如“AutoOpen”）存在，如果有这样的文件，Word 就启动它。同样，假如有“AutoClose”宏存在，则 Word 在关闭一个文件时会自动执行它。通常，宏病毒至少包含一个以上的“自动宏”，当 Word 运行这类自动宏时，实际上就是在运行宏病毒代码。宏病毒的内部都具有把带病毒的宏复制到通用宏的代码，也就是说，当病毒代码被执行过后，它就会将自身复制到通用宏集合内。当 Word 系统退出时，会自动把所有通用宏（包括传染进来的病毒宏）保存到模板文件（Normal.dot）中，这样，一旦 Word 遭受感染，则以后每当 Word 进行初始化时，系统都会随着 Normal.dot 的装入而成为带病

毒的 Word 系统，继而在打开和创建任何文件时感染该文档。感染 Normal.dot 模板是宏病毒最常见的传染方式。此外，与计算机系统启动相似，Word 在启动过程中会执行 Office/StartUp 目录中的模板文件（扩展名为.dot）所包含的宏。有些病毒就是通过这两种方式来感染 Word 系统，使每次启动后的 Word 都成为带毒的运行环境。实际上，凡是具有宏的其他软件，包括 Excel、PowerPoint、Access、WordPad 都可能受到这种病毒的感染。

下面是对宏病毒 Word.Cyber-Hacker 一些代码片段的分析，以此了解宏病毒的机理。

```
Sub AutoOpen
    Call Cyber                                    //每次打开文档时，将病毒传染到模板中//
End Sub
Sub Cyber（）
     Call CyInit                                          //为病毒传染做初始化工作//
     Call Dok2Nor                                  //将当前文档中的宏病毒传染到模板中//
     Call CyClose                                         //恢复 Word 原有的参数值//
End Sub
Sub CyInit（）
     …
     CommandBars（“Visual Basic”）.Visible=False      //关闭 Visual Basic 工具条的显示//
     CommandBars（“Visual Basic”）.Protection=msoBarNoChangeVisible
                                                          //使该工具条不能被改变为显示//
     CommandBars（“Visual Basic”）.Protection=msoBarNoCustomize
                                                              //防止用户“自定义”工具栏//
          …
          FindKey（BuildKeyCode（wdKeyF11,wdKeyAlt）.Disable
//关闭 Alt、F1 等功能键，不许通过快捷键执行“宏”命令//
          …
End Sub
…
Sub Dok2Nor（）
     …
Set ad=ActiveDocument                             //在 ad 中记下当前活动文档//
     Set nt=NormalTemplate                                    //记下 Normal.dot//
…
                                        //下面将当前打开文档中的宏病毒传染到模板中//
Application.OrganizerCopy Source:=ad.FullName,_
     Destination:=nt.FullName,Name:=_
     “CyberHack”,Object:=wdOrganizerObjectProjectItems
Application.OrganizerCopy Source:=ad.FullName,_
     Destination:=nt.FullName,Name:=_
     “CyberForm”,Object:=wdOrganizerObjectProjectItems
Templates（nt.FullName）.Save                  //保存已感染的 Normal.dot 模板文件//
…
     End Sub
```

3．网络病毒

（1）网络病毒的特点

根据目前的病毒发展趋势，网络病毒越来越流行，它一般具有以下特点。

1）主要通过网络传播，在网络环境中才能发挥最大破坏作用。例如，“木马”类病毒离开网络后，它最大的危害仅仅是消耗一点系统资源而已。

2）寄生宿主广泛，可能会寄生在 HTM、ASP 等文件中；也可能隐藏在邮件中；甚至可能不感染任何对象，仅存于源宿主中，但可通过网络传播对计算机的端口、服务、数据、缓冲区进行攻击的指令。所以网络病毒的检测、防范难度很大。

3）一般是利用 Internet 的开放性、操作系统及应用程序的漏洞来对计算机系统进行攻击，为了防范它，往往要对某些网络功能进行限制。

4）一些网络病毒还常常与黑客有联系，最典型的就是“木马”类病毒。

5）网络病毒发展趋势迅猛，近几年来流行的病毒，除了宏病毒外，基本都属于网络病毒。可见，网络病毒的传播速度快、危害范围广。

（2）网络病毒的种类

网络病毒有很多种，如脚本病毒、邮件病毒等。依据病毒的攻击手段，可将网络型病毒分为蠕虫和木马两大类型。

1）蠕虫。蠕虫（Worm）是通过分布式网络来扩散传播特定信息或错误，破坏网络中的信息或造成网络服务中毒的病毒。蠕虫泛滥发生在近几年，但早在 1982 年，Shock 和 Hupp 就提出了一种“蠕虫”程序的思想，这种“蠕虫”程序常驻于一台或多台机器中，并有自动重新定位的能力。如果它检测到网络中的某台机器未被占用，就把自身的一个副本（一个程序段）发送给那台机器。早期的“蠕虫”程序不一定是有害的，它可以作以太网网络设备的诊断工具。“蠕虫”一般由两部分组成：一个主程序和一个引导程序。主程序一旦在机器上执行，就会通过读取公共配置文件及收集当前网络状态信息，获得与当前机器联网的其他机器的信息和软件缺陷，主动尝试利用所获得的信息及其他机器的缺陷在这些远程机器上建立其引导程序。

蠕虫病毒最主要的特点是利用网络中软件系统的缺陷，进行自我复制和主动传播。2003 年 1 月 25 日首次发作的 Win32.SQLExp.Worm 病毒就是一个非常典型的蠕虫病毒，它具备了蠕虫病毒所有的典型特征。

2）木马。木马又称特洛伊木马（Trojan Horse），它原本属于一类基于远程控制的工具。木马的运行模式属于 C/S 模式，它包括两大部分，即客户端和服务端。其原理是一台主机提供服务（服务器），另一台主机接收服务（客户端）。作为服务器的主机一般会打开一个默认的端口进行监听。如果有客户端向服务器的这一端口提供连接请求，服务器上的相应程序就会自动运行，来应答客户机的请求。这个程序被称为守护进程。木马通常的攻击步骤如下。

- 设定好服务器程序。
- 骗取对方执行服务器程序。
- 寻找对方的地址 IP。
- 用客户端程序来控制对方的计算机。

木马之所以能够运行是由于木马的程序通常继承了与用户相同、唯一的优先权和存取权。它能够在不触犯系统安全规则的情况下进行非法活动，系统本身不能区分木马和合法程序。通常所说的木马病毒其实就是这个服务端程序，它通过电子邮件或网页传播到用户的计算机中，一旦计算机执行这段程序，它就变成一个受客户端控制的服务器。木马常常被作为黑客用来窃取信息及非法使用资源的工具。

（3）网络病毒的传播方式

网络病毒的传播方式主要有 3 种：电子邮件、网页、文件传输。其中主要通过前两种方式，特别是电子邮件。通过电子邮件传播的病毒，其病毒体一般隐藏在邮件附件中，只要执行附件，病毒就可能发作。有些种类的邮件病毒，甚至没有附件，病毒体就隐藏在邮件中，只要打开或预览邮件，都会遇到麻烦。近年来流行的很多病毒，如“梅丽莎”、“爱虫”等都是通过邮件传播的。为了增加网页的交互性、可视性，通常需要在网页中加入某些 Java 程序或者 ActiveX 组件，这些程序正是病毒的宿主。如果浏览了包含病毒代码的网页，且浏览器未限制 Java 或 ActiveX 的执行，其结果就相当于执行了病毒程序。文件传输的传播方式主要是指病毒搜寻网络共享目录，把病毒体复制入其中，远程执行或骗取用户执行。

（4）脚本语言

网络病毒的编写者很热衷于使用脚本，特别是使用 VBScript 脚本编写病毒代码。VBScript 又通过 Windows Script Host 来解释执行。一个脚本程序能调用功能更大的组件来完成自己的功能。病毒还可能将代码自动加入到附件中发送到网络，实现病毒的主动传播。脚本病毒类似于前面所描述的宏病毒，但是它的执行环境不再局限于 Word、Excel 等 Microsoft Office 应用程序，而是随着 Microsoft 将脚本语言和 Windows 操作系统日益紧密的结合，扩展到网页、HTA（基于 HTML 的应用程序），甚至文本文件中。脚本语言是介于 HTML 和 Java、C++和 Visual Basic 之类的编程语言之间的语言。HTML 通常用于格式化文本和链接网页，基本上没有处理功能；编程语言通常用于表示一系列复杂指令和逻辑。脚本语言也可用来向计算机发送指令，但它们的语法和规则没有可编译的编程语言那样严格和复杂，而且脚本语言是解释执行的，可以直接执行脚本语言的文本，而不需要使用一个庞大的编译器将脚本语言编译成机器语言。执行脚本语言需要一个脚本语言引擎解释执行脚本语言编程的程序。主要的脚本语言包括以下几种。

- 活动服务器页面（Active Server Pages）：在 Internet 服务器执行的脚本，构造变化多端的页面供浏览。
- 微软可视化 Basic 脚本语言（Microsoft Visual Basic Scripting Edition），使用 Microsoft Windows 操作系统内置的脚本引擎。
- Java 脚本语言（Java Script）：使用浏览器内置的脚本引擎执行。
- 其他的脚本语言，如 PHP、REXX、PERL 等。

由于能自我复制、扩散的程序就可以成为病毒，且脚本语言的功能越来越强大，现代的脚本语言基本上可以完成所有的文件系统操作。所以使用脚本语言的病毒的出现也就成为必然。

脚本语言存在的一个最重要的基础和前提是，如何满足病毒的基本前提，即复制自身。使用 VBScript 可以很容易地完成这个任务。下面是一个普通的 VBScript 脚本（使用伪代码）：

```
设置　　对象 1=创建对象（“Scripting.FileSystemObject”）//创建一个文件操作对象
```

对象 1.创建文本文件（“virus.txt”，1）//通过这个文件对象的方法创建一个 TXT 文件

如果把这两句保存成为扩展名.vbs 的 VBScript 脚本文件，单击时就会在当前目录创建一个文本文件。如果把第二句改为

对象 1.获取文件（“WScript.ScriptFullName）.复制（“virus.vbs”）

就可以将自身复制到当前目录的 virus.vbs 文件中。它的意思是把程序本身的内容复制到目的位置。这么简单的两行代码就实现了自我复制的功能，已经具备病毒的基本特征。

图 5-4 所示为邮件病毒脚本，可以实现对 Microsoft 邮件系统的传染。只要具有一个脚本病毒的基本功能，这些代码就可以通过 Outlook 的邮件系统把自己以附件的方式扩散出去。

从这个简单的伪代码中可以看出，仅仅这么简单的几行代码，就能成为具有自我复制、繁殖、骚扰网络的病毒了。当然，我们可以为它添加更多的本领，如修改注册表、删除文件、发送被感染者的文件、隐藏自己、感染其他文件等。其实，以目前 Windows 系统的编程开放特性及脚本语言和 Windows 操作系统的紧密集成，上面的功能都很容易实现。一个中级的 VB 程序员，没有任何的汇编知识，很容易就能制作类似的蠕虫病毒。

脚本病毒包括下面几种基本类型。

1）基于 JavaScript 的脚本病毒。使用 JavaScript 语言编写的病毒，主要运行在 IE 浏览器环境中，可以对浏览器的设置进行修改，主要的破坏是对注册表的修改，危害不是很大。

2）基于 VBScript 的脚本病毒。使用 VBScript 语言编写的病毒，可以在浏览器环境中运行。更重要的是，这种病毒和普通的宏病毒并没有非常清晰的界限，可以在 Office（主要是 Outlook）中运行，可以执行的操作非常多。前面描述的例子就是使用 VBScript 语言编写的。此类病毒还可以修改硬盘上的东西、删除文件、执行程序等，危害非常大。

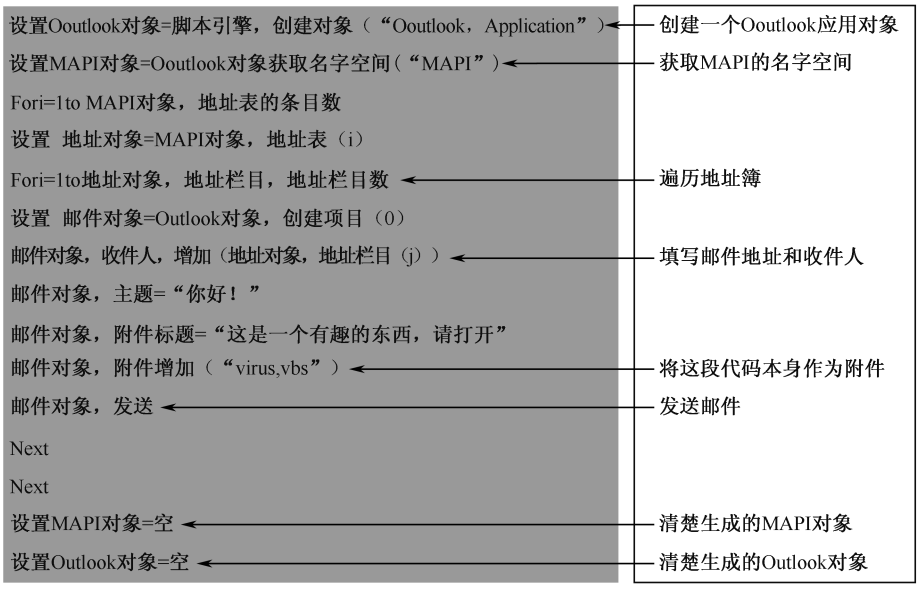


图 5-4　邮件病毒脚本

3）基于 PHP 的脚本病毒。这是新的病毒类型，可感染 PHP 脚本文件，主要对服务器造成影响，对 PC 影响不大。目前，仅有一个“新世界”（New World）病毒存在，并没有造成很大的破坏。但是其前景非常难以估计，如果 PHP 得到更加广泛的使用，这种病毒将成为真正的危险。

4）脚本语言和木马程序结合的病毒。这种病毒除了使用脚本语言进行扩散以外，还会在受到入侵的计算机上安装一个名为特洛伊木马的程序，容许他人未经授权访问受到感染的计算机。一种典型的方法是，通过直接在病毒代码（包括二进制的木马程序）中编码，或者另外一个称为 virus.bin 的文件，通过脚本语言直接执行 DEBUG 程序，使用 DEBUG 程序将 virus.bin 存储成为 virus.exe，然后通过脚本语言就可以偷偷地执行这个木马程序。这种木马和脚本相结合的病毒已经成为最近病毒发展的新趋势，也给反病毒软件厂商带来了很大的挑战。

小结

综上所述，科技的日新月异对网络安全也带来了前所未有的挑战，维护网络安全的任务越来越艰巨。唯有提高对病毒的认知能力、通过防火墙、信息加密等的层层防护才能在一定程度上确保网络信息的安全传输。

第 6 章 通信网

6.1 固定电话网

固定电话网又可被称为公用电话交换网（Public Switched Telephone Network, PSTN），是采用固定终端的一种业务网络，可以分为本地网和长途网。本地网是指在统一编号区域范围内，由若干个端局（即交换中心）或者若干个端局和汇接局所组成的电话网。长途网是指在全国范围内不同编号区之间通话的网络，由端局、汇接局和若干个长途局所组成的电话网。从设备使用量和业务量来考察，固定电话网仍是目前最大的电信业务网之一。

6.1.1 固定电话网的起源

电话通信是日常生活中最常见的一种通信方式，使用率最高，应用也最广泛，它将声音信号转化为电压信号通过电话线传送到另一端，再利用送话器转化为声音信号。

1876 年美国科学家贝尔发明了电话机，最初的电话通信能够完成的任务，仅限于两部电话机间固定通信，即两部电话机间的点对点通信，如**错误!未找到引用源。**所示。

点对点通信存在以下缺点。

- 点对点传输网络中，任意两个终端之间都需要一条专门的传输线路。当终端个数为 N 时，传输线路数为 $N(N-1)/2$ 对，即当终端数增加时，传输线路数也会急剧增加，网络规模变得庞大复杂，如图 6-2 所示。

图 6-1　点对点通信

- 每个终端都需要 $N-1$ 个线路接口，接口过多。
- 每增加一个终端 N，就需要增设 $N-1$ 对线路。

- 终端距离越远，信号传输损耗越大。

交换的设想由美国人阿尔蒙 • B．史端乔在 1878 年提出。交换的基本思想是将多个终端通过一个转接设备互连，通过转接设备传递任意两台终端中的信息，如图 6-2 所示为多个终端的点对点通信。利用交换设备的通信如图 6-3 所示，这个转接设备被称为交换机。交换机既降低了线路投资，又提高了线路的利用率。

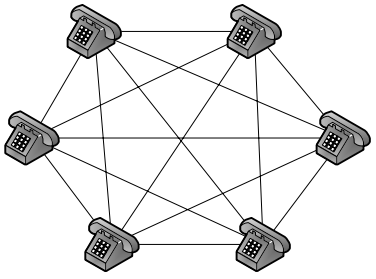


图 6-2　多个终端的点对点通信

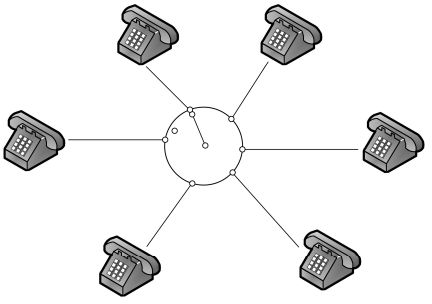


图 6-3　利用交换设备的通信

6.1.2　固定电话网的组成

固定电话网由交换机、传输设备和终端设备 3 部分组成的，如图 6-4 所示。工作时先由电话机的送话器把话音信号转换为电信号，经适当处理（滤波、放大等）由传输线路送到交换机，然后根据主叫用户的拨号接通被叫用户的线路，把信号转接到对方的受话器，受话器把电信号还原成话音。

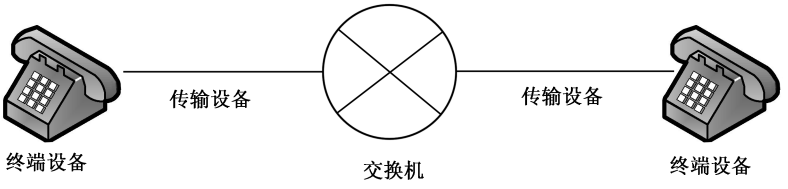


图 6-4　固定电话网的组成

（1）终端设备

终端设备一般意义上指的就是电话机。电话机的基本功能是在电信号与话音信号之间的相互转换。用户在说话时，电话机将话音信号转化为电信号；接听时，电话机将电信号转化为话音信号。随着业务的发展，用户终端设备也在不断地发展之中，现在已经发展成为包括传真机、计算机、数据发信机等在内的各种设备。

（2）传输设备

传输设备是指终端设备与端局，或端局与端局之间的传输线路及其相关设备。传输设备既可以传输模拟信号，又可以传输数字电信号。用户终端与端局之间的传输介质以双绞线为主，端局与端局之间采用光纤连接。

（3）交换机

交换机是网络中所有节点的汇聚点，它根据网络中的地址信息和信令信息连通不同的节点，使固定电话网络中的用户建立连接实现通话。交换机经历了人工交换、时分交换、空分交换等，直到数字程控交换机、ATM 交换机等过程。交换机的基本功能如下。

- 呼叫检查功能。
- 接收被叫号码。
- 若被叫空闲，则应分配空闲的通话回路。
- 向被叫用户振铃，向主叫送回铃音。
- 被叫应答后，接通话路。
- 及时发现话终，进行拆线，并复原话路。

为了完成上述基本功能，交换机必须具有话路系统和控制系统。话路系统具有进行通话的功能，为了实现该功能，话路系统应具有用户电路、交换设备、出入中继器、绳路和信号设备等。控制系统用来连接话路，为了实现该功能，控制系统应具有译码、忙闲测试、路由选择、链路选试、驱动控制和计费等设备。

6.1.3 固定电话网的特点

固定电话网的设计和目标是支持话音通信，它具有以下特点。

（1）同步时分复用

固定电话网中一般采用的是同步时分复用技术，其原理是将多个用户的信息通过同一条物理媒介传输，不同的用户之间通过时分复用的方式共享这一条物理信道，通过这种方式可以提高线路的利用率。同步时分复用中每个用户占用一个时隙，而且是固定的时隙，多个用户的时隙组成一帧，因此，每个用户占用的带宽是固定的，故此语音通信的速率也是恒定的。

（2）同步时分交换

要想实现两个用户之间的通信，就必须将这两个用户的信息进行交互。将两个用户所在时隙的信息同时交互，就可以实现两个用户之间的通信，这一方法称为同步时分交换。

（3）面向连接

当用户进行呼叫时，固定电话网络为两个用户之间建立一条专享的链路。当被叫用户接通时，双方可以直接进行信息的交互，建立一条面向连接的通路。

（4）对用户数据透明传输

由于固定电话网络为相互通信的用户之间建立一条专享的通信链路，用户数据可以直接在上面传输，而不需要添加冗余控制信息，因此固定电话网络中用户数据可以透明地在网络中进行传输。这种传输方式网络时延小，通话质量高，适合于语音通信。

6.1.4 固定电话网组网

很多国家的固定电话网采用等级结构。等级结构顾名思义就是把电话网划分为不同的等级，

低级交换中心与控制它的高级交换中心连接，形成星型网。最高级的交换中心直接互连，形成网状网。所以，电话网一般为复合型网络。

影响等级结构级数选择的因素有很多，其中有两个主要因素。

- 全国固定电话网的服务质量，如可靠性、传输时延、传输质量和接通率等。
- 全国固定电话网的成本问题，即搭建网络的费用问题。

除此之外，还应当考虑国家的地理面积大小，以及各个地区不同的地理状况、经济、政治条件等因素。

我国的固定电话网也采用了等级制，共有 5 级，如图 6-5 所示。

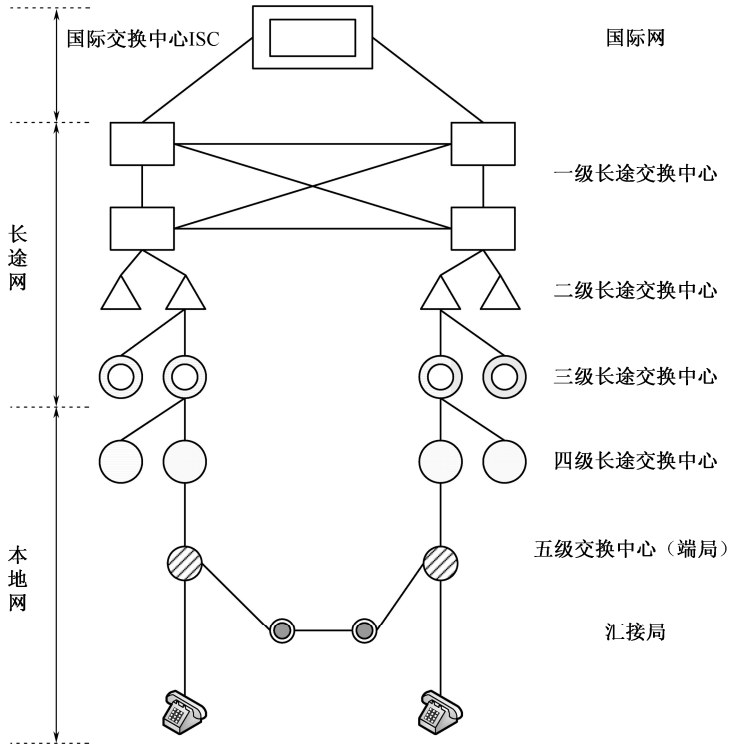


图 6-5　我国固定电话网的 5 级结构

我国固定电话网的 5 级结构依次为：一级长途交换中心（C1）、二级长途交换中心（C2）、三级长途交换中心（C3）、四级长途交换中心（C4）和端局（与用户相连接的交换局 C5）。而长途电话网就由其中的一级～四级长途交换中心（C1~C4）构成，一级长途交换中心进行互连可以形成网状型网络。端局、用户与本地汇接局组成了本地电话网。

由于固定电话网的业务量快速增加，用户对于新业务的要求也越来越多，而现有的 5 级网络结构由于转接段数多造成的接续时延长、传输损耗大、网络管理工作复杂等问题，已逐渐不能满足用户和运营商的需求，所以我国正在逐步把 5 级电话网转变为 3 级电话网。

（1）本地电话网

本地电话网由端局、汇接局、局间中继线、长市中继线、用户线及电话机等组成，且必须

在一个长途编号区的服务范围内。

（2）端局

本地网中的端局（DL）只有本局交换功能和来、去电功能，并且直接与用户相连。在某些情况下，端局还会与用户模块、用户集线器、PABX 等用户装备连接。

由于端局设置地点不完全一样，可以把端局分为市内端局，县（市）及卫星城镇端局，农村乡、镇端局。

（3）汇接局

汇接局（Tm）就是本地网中负责转接端局话务或汇接端局与长途局的交换中心。还有一些汇接局拥有疏通用户的来、去电功能，与端局的功能相似，这一汇接局称为混合汇接局。

网络分为网状网与二级网两种，这样建设网络是由于各城市的经济状况、政治地位及人口不同，网络规模相差较大，本地网交换设备的扩容量也不同。

（4）网状网

本地网络结构中最简单的结构是网状网，如**错误!未找到引用源。**所示。从图中可以看出，网络中所有终端互相连接，终端之间有直达电路。当本地网络内交换终端数较少时，便可以采用这种网络结构。

（5）二级网

当本地网中交换终端数较多时，便可以使用二级网。终端和汇接局形成两级结构的等级网络，终端作为第一级，汇接局作为高一级。二级网络的结构有两种：分区汇接、全覆盖。

分区汇接网络是将本地网络分为很多个汇接区，并选择汇接区内话务密度大的一或两个局作为汇接局，如图 6-7 所示。

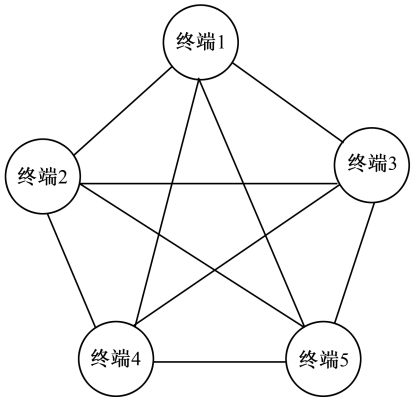


图 6-6　网状网结构

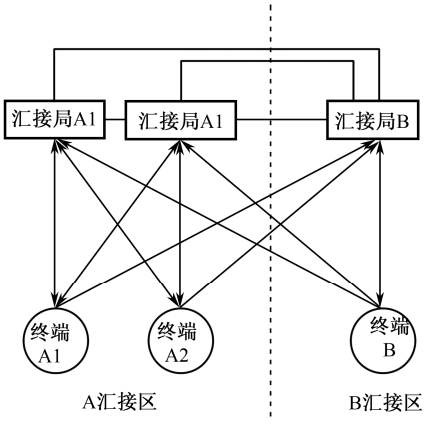


图 6-7　本地网的分区汇接局结构

全覆盖网络结构是在本地网络中建立多个汇接局，各个汇接局平均分担话务负荷，地位平等。汇接局之间用网状网互相连接，两端局间的用户进行通话最多转接一次，如图 6-8 所示。

全覆盖网络的网络结构适用范围很广，各种规模和类型的本地网基本都适用。汇接局的树木结构可以根据网络的规模来确定。全覆盖结构可靠性很高，但是线路成本也较高，所以确定网络结构时应综合考虑这两个因素。

（6）长途电话网

国内长途交换中心分为省级长途交换中心和本地网长途终端交换中心，省级长途交换中心称为一级交换中心，负责汇接全省转接长途话务和一级交换中心所在本地网的长途终端话务；本地网长途终端交换中心称为二级交换中心，负责汇接本地网长途终端话务。其网络结构如图 6-9 所示。

根据话务流量流向，二级交换中心也可与非从属一级交换中心建立直达电路群。

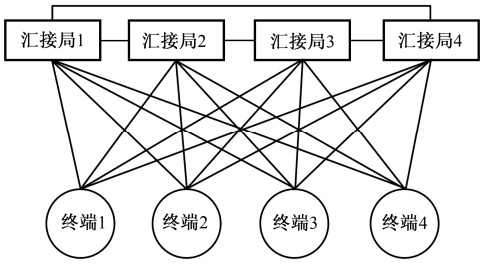


图 6-8　本地网的全覆盖网络结构

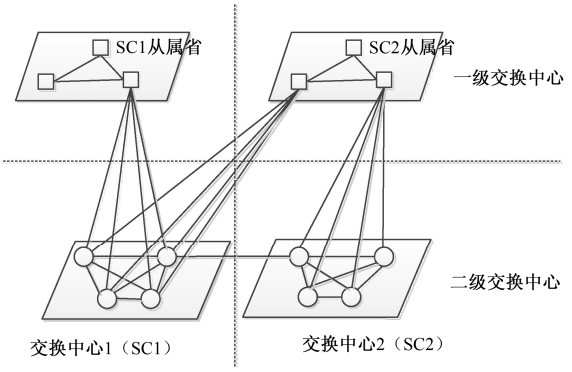


图 6-9　二级长途电话网网络结构

6.1.5　固定电话网编号

电话号码是固定电话网中电话用户的唯一标识，也是电话交换机对路由的选择依据。因此在任何一个电话网中，用户号码都应当按照一定的规律按计划编号，不可重复。

（1）本地网用户号码的组成

本地网用户号码由“局号+用户号码”的方式组成。

（2）国内长途直拨用户号码的组成

国内长途直拨用户号码由“长途字冠+长途区号+被叫号码”组成。

- 长途字冠：我国长途全自动接续字冠为“0”。
- 长途区号：我国长途区号按不等位原则编制，以防号码过长。
- 被叫号码：被叫用户在本地网中的编号，统一采用等位编号。

（3）特种业务编号

特种业务号码按 3 位编号，首位是“1”。

（4）国际长途直拨号码的组成

国际长途直拨号码由“国际长途字冠+国家号码+该国国内号码”组成。

- 国际长途字冠： 它是国际电话的标志，由各国自行规定，我国规定为“00”。呼叫国际长途时，国内长途局根据国际长途字冠识别后，接入国际电话网。
- 国家号码：统一规定为 1～3 位。我国为“86”，长度为两位。

（5）国内长途区号的编制原则

- 北京：长途区号长两位，编号为 10。
- 大区中心：长途区号长两位，第一位为 2，第二位在 1～9 中取数。
- 省中心：长途区号长三位，第一位在 3～9 中取数，第二位取奇数，第三位在 0～9 中取数。
- 地、市中心：同省中心。
- 县中心：长途区号长四位，第一位在 3～9 中取数，第二位取偶数，第三位和第四位在 0～9 中取数。

6.2 X.25 分组网

X.25 是一种广泛应用的协议，是 20 世纪 70 年代由国际电报电话咨询委员会（CCITT）制定的“在公用数据网上以分组方式工作的数据终端设备（DTE）和数据电路设备（DCE）之间的接口”。X.25 的协议集有 3 层，与 OSI 协议栈的底三层相关联，即物理层、数据链路层和网络层。

X.25 协议是 CCITT 建议的一种协议，它建立了终端和分组交换网络之间的连接。多个不同的用户可以通过 X.25 协议连接到远程主机上，并通过分组交换网络选择目的路由，以实现任意用户之间的连接。

基于 X.25 协议的分组交换体系结构也有着它自己的优缺点。X.25 协议中信息分组根据分组头中的目的地址来选择通过散列网络的路由，这样用户可以到达任意目的地，但是由于用户要进行路由选择，信息传输就存在一定的延迟。而且，分组网络中多个用户共享同一链路，随着用户的增加，每个用户的带宽减少，用户速率降低。面向连接的电路交换中通话的两点之间拥有一条专用线路，用户通话的带宽固定，速率稳定，信息基本没有延迟。

在 X.25 中，一个传输路径上的每一个节点都必须完整地接收一个分组，并且完成检查后才能发送出去。而帧中继节点只查看分组头中的目的地址就开始转发这个分组，有的时候这个分组还没有完全接收就已经开始转发。帧中继中没有用于处理管理、流控和错误检查的状态表，所以 X.25 的开销比帧中继要高许多。

X.25 虽然因为低性能而不能用于实时 LAN 对 LAN 应用，但是 X.25 容易建立与理解，可以被远程终端或计算机访问，并且可以在传输量较低的情况下采用。X.25 可以为一些话务量小的国家提供网络连接。

X.25 协议对应 OSI 协议的物理层、数据链路层和分组层，但是也有些不同。

1）物理层。在 X.25 协议中，物理层称为 X.21 接口，定义它是从终端到 X.25 分组交换网络中的附件节点的物理/电气接口。RS-232C 通常用于 X.21 接口。

2）数据链路层。在 X.25 协议中，数据链路层又称帧层，它采用链路接入平衡方式，是高级数据链路控制（HDLC）协议的一部分。

3）分组层。在 X.25 协议中定义了通过分组交换网络的可靠虚电路，提供了点对点数据发送，而不是一点对多点发送。

X.25 的层次结构如图 6-10 所示。

虚电路在 X.25 协议中非常重要。两个用户之间需要进行信息传输时，可以在它们两者的分组层建立一条虚电路，即两者之间可以建立一条“逻辑”信道，为数据在两者之间提供了可靠的传输方式。

在 X.25 中有两种类型的虚电路。

- 临时性虚电路：将建立基于呼叫的虚电路，然后在数据传输会话结束时拆除。
- 永久虚电路：在两个端点的节点之间保持一种固定连接。

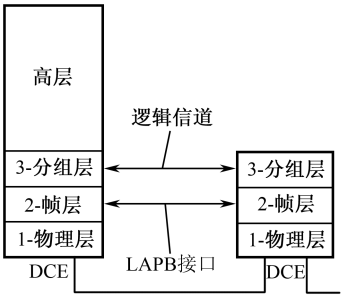


图 6-10　X.25 的层次结构图

6.3　数字数据网 DDN

数字数据网（Digital Data Network，DDN）利用数字信道传输数据信号，传输质量高、速度快、带宽利用率高。卫星通道、数字微波、光缆和双绞线都是它的传输媒介。

1．DDN 网的特点

（1）全透明传输

DDN 数字通信的协议和规程通过智能化用户终端完成。DDN 是全透明网，面向各类数据用户，不受任何规则约束，是一种公用信息网。

（2）数据同步

DDN 采用的数据传输方式是数字传输，所以必须保持系统同步才可以得到低误码率、高质量的传输。否则，网络内电路的转接、分支时及电路互连时将很难协调工作，甚至会出现失步，造成数据的重复或定期挂失的现象。

（3）高速传输，网络延时小

DDN 用户根据事先约定的协议，采用时分复用技术，固定通道带宽和预先约定速率下顺序连续传输，免去了目的终端对信息的重组，减少了延时。

（4）灵活的连接方式

DDN 网可以支持多种业务，为用户提供灵活的组网环境。其不仅可以和用户网络连接，也可以和用户终端设备进行连接。

（5）电路可靠性高

DDN 采用路由迂回和备用方式，使电路安全可靠。

（6）保密性高

DDN 专线提供点到点的通信，信道固定分配，通信保密性强，保证通信的可靠性，不会受其他客户使用情况的影响，因此特别适合金融、包厢客户的需要。

（7）网络运行管理简便

DDN 网采用网络管理对网络业务进行调度监控，减少了不必要的人为错误，使网络管理智能化。

总之，DDN 提供了高速、高质的通信环境，为用户规划、建立自己安全、高效的专用数据网络提供了条件。

2．DDN 网的结构

我国 DDN 网络规模大、数量多，为了组网灵活、扩容方便、业务组织管理清晰，按网络功能层次把 DDN 划分为核心层、接入层和用户接口层 3 层网络结构，如图 6-11 所示。

核心层由大、中容量网络设备组成，用 2048kb/s 或更高速率的数字电路互连；接入层由中、小容量网络设备组成，用 2048kb/s 的数字电路与核心层互连，并为各类 DDN 业务提供接入；用户接口层由各种用户所用设备、网桥/路由器设备、帧中继业务的帧装/拆设备组成，也可以在用户接口层设置小容量网络设备，提供子速率复用、模拟话音/G3 传真业务的接入。

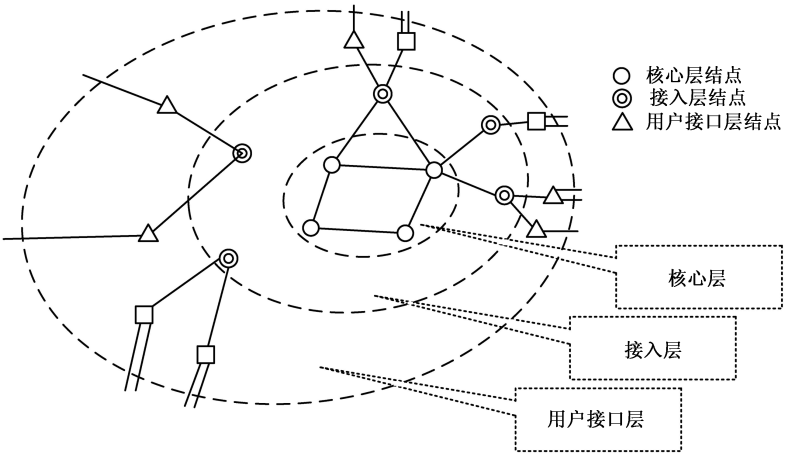


图 6-11　DDN 网的分层结构

3．DDN 网的组成

如图 6-12 所示，一个 DDN 主要由交叉连接和复用系统、本地传输系统、网络同步系统、局间传输及同步时钟系统和网络管理系统这五大部分组成。

（1）本地传输系统

本地传输系统指的是用户线路，即 DDN 的本地局到终端用户之间的传输系统，一般采用普通的市话用户线，也可使用电话线上复用的数据设备。

（2）交叉连接和复用系统

复用是将多个用户的数据流按时分复用的原理合成 64kb/s 的数据流，通常称为零次群号（DS0），然后再将多个 DS0 信号复用成一次群，即 2.048Mb/s 或更高次信号。交叉连接是由网关中心的操作员将符合一定格式的用户数据信号与零次群复用器的输入或者将一个复用器的输出与另一复用器的输入交叉连接起来，实现半永久性的固定连接。

（3）局间传输系统

局间传输系统是指各节点根据信息流量、流向等因素组成的网络拓扑，各节点以 2048kb/s 的数据速率进行数字通信。

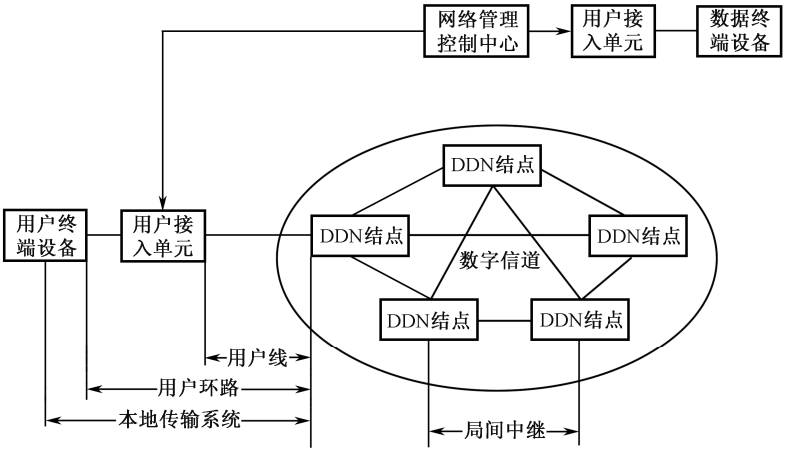


图 6-12　DDN 网组成的结构示意图

（4）网络时钟同步系统

网络时钟同步系统保证 DDN 全网设备的同步工作，可以采用主从同步方式，也可采用多颗卫星覆盖的全球定位系统（GPS）来实施。

（5）网络管理系统

DDN 的网络管理包括网络资源的调度、网络状态的监控、用户接入管理、路由选择、网络故障的诊断、网络运行数据的收集与统计、告警与处理、计费信息的收集与统计等。对于大范围的公用 DDN，网络管理采用分级管理的方式。

4．DDN 网的应用

DDN 为用户提供的基本业务是专用电路业务，其中最典型的是点对点的 DDN 专线。DDN 专线和公用电话交换网中的电话专线不同，DDN 专线的连接是半固定连接，是一个数字信道，其质量高、带宽大，具有较完善的网络管理系统。电话专线的连接是固定的物理连接，且为模拟信道，带宽小、质量差，没有完善的网络管理系统。DDN 连接信道的数据传输速率、路由及所用的网络协议等可随时根据需要申请改变，以充分满足用户的通信要求和通信质量。其应用范围如下。

- 飞机票、火车票出售网。
- 信息数据库查询系统。
- 股市行情广播及交易。
- 银行联网。
- 数据传输、图像传输、语音传输。

5．DDN 网与 X.25 网的比较

DDN 与 X.25 网的区别在于：X.25 是一个分组交换网，具有协议转换、速度匹配功能，适用于不同通信规程、不同通信速率的用户设备之间的通信。它本身具有 3 层协议，用户通过呼叫建立虚电路。而 DDN 不具备交换功能，主要方式是租用专线，用户申请专线后，连接就已完成。在线路透明度方面，DDN 是一个完全透明的网络，在用户速度小于 64kb/s 时，采用子速率复用技术，大于 64kb/s 时，采用时分复用技术；而 X.25 只有在高层协议上对用户透明。在收费方面，DDN 按固定月租收费，而 X.25 按通信字段（流量）收费。所以 DDN 更适合于需频繁通信的 LAN 与 LAN 或主机的互连。

6.4 帧中继 FR 网

帧中继网（Frame Relay，FR）是在用户—网络接口之间提供以帧为单位的按顺序进行用户信息流的双向传送，吞吐量高，适合突发性业务的网络。帧中继对用户信息流进行统计复用，“帧”的容量比分组交换网的分组容量大得多；而且帧中继仅完成 OSI 物理层和数据链路层核心层的功能，将流量控制、纠错等留给智能终端去完成，大大简化了节点之间的协议，应用更为简便。

1．帧中继网的特点

（1）网络费用低廉

当使用专用帧中继网时，可以按照需要分配带宽，并把多个用户的数据复用到主干网络上，提高网络的利用率，降低系统成本。

（2）数据传输效率高

帧中继采用同步时分复用技术，可以使多个用户共享同一网络，提高了系统资源的利用率；同时，帧中继节点间的协议比较简单，因而通信时延小，速率高。

（3）数据传输可靠

帧中继虽然没有纠错和流量控制功能，但是由于其采用光纤进行传输，信号质量较好，信道错误率低，而且终端智能化程度较好，可以进行端到端的修复。

（4）连接方式灵活

帧中继具有协议简单、操作简便和组网灵活的特点，使得其可以很容易地应用到现有网络上。而且，帧中继对高层协议是透明的，可以兼容各种协议，使用户很方便地接入，并且适用于各种业务类型。

2．帧中继网的结构

帧中继协议模型包括物理层和数据链路层核心功能两层，协议模型如图 6-13 所示。

3．帧中继网的应用

帧中继主要作为分组交换网络的中继传输网络，其使用范围包括如下方面。

- 大企业、银行、政府部门总部和各地分支机构的局域网之间的互连。
- 数据的检索。
- 大文件的传送。
- 组建虚拟专网。

4．帧中继网与 X.25 网的比较

帧中继可以看做 X.25 的部分主要功能在新的传输条下的发展，由于其协议比较简单，开销较小，时延也小，数据传输效率比较高，因而可以与 X.25、局域网和广域网连接，为其提供大容量的数据传输。

帧中继使用 X.25 数据链路层 HDLC 的帧格式，但是不采用 LAPB 规程，而是采用 LAPD 规程。LAPD 规程是按照 ISDN 标准制定的，更适用于独立的信令控制信道。帧中继与 X.25 如图 6-14 所示。

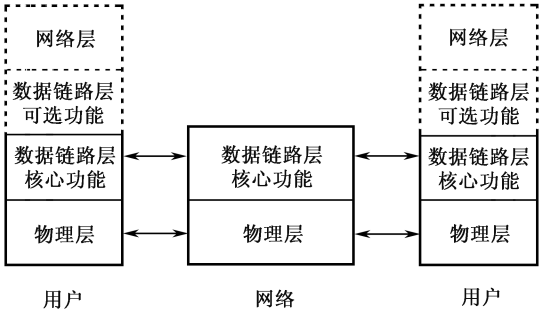


图 6-13　帧中继的协议模型

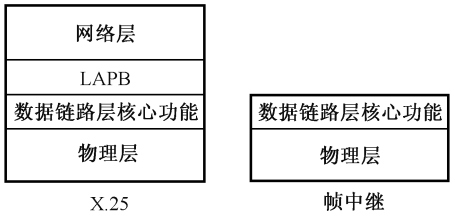


图 6-14　帧中继与 X.25

6.5　VoIP 技术

宽带电话（Voice over IP，VoIP）是在欧美等发达国家应用比较成熟的一种新型通信方式，它以宽带为载体来实现语音通信，这种方式在我国尚处于起步阶段。目前，VoIP 主要通过 Internet 实现宽带语音通信，该领域涉及技术较多，实现非常复杂。

1．VoIP 简介

VoIP 技术主要通过对语音信号进行数字化编码和压缩等一系列处理，最后转化为 IP 数据包在 IP 网络上传输，以实现语音通信宽带化。在固定电话网络中，每路语音信息占用 64kb/s 的带宽，采用 VoIP 技术后，语音信息被压缩至只占用 1/3 的带宽，通过这种方式提高了网络的利用率，降低了通信资费，并且可以有效地解决网络拥塞的问题，提高了服务质量。VoIP 技术

可以在 IP 网络上传输比固定电话网络更多的业务，如可以传输语音、传真、视频和数据等业务，提供更好的服务，促进宽带多媒体的发展。

（1）VoIP 的基本传输过程

VoIP 技术是对模拟的语音信号进行压缩、打包等一系列的特殊处理，采用无连接的 UDP，使之可以在 IP 分组网络上进行传输的技术。

为了使语音信号可以在 IP 网络中传输，要求网络中具备 VoIP 功能的设备。VoIP 系统的基本构成如图 6-15 所示。

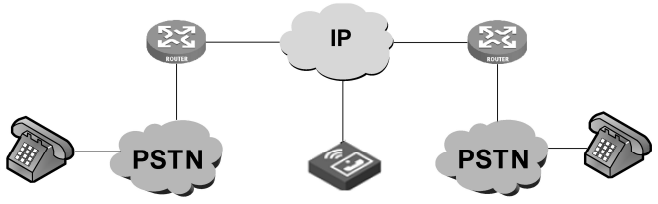


图 6-15　VoIP 系统的基本构成

从图 6-15 中可以看出，用户直接接在公用电话网（PSTN/ISDN）上，通过公用电话网连通 IP 语音网关，将模拟语音信号转化为数字语音信号，再通过 IP 网络传送至被叫用户所在的 IP 语音网关，将数字语音信号转化为模拟语音信号，并通过公用电话网传送至被叫用户。

VoIP 的传输过程可分为以下几个阶段。

1）语音-数据转换。

语音信号要在 IP 网络上进行传输，需要先将模拟信号量化编码变为数字信号。量化编码过程中，语音包长度通常为 60、120 或 240ms，量化编码后的数据帧长通常为 10～30ms。经常采用的语音编码方式有 ITU-T G.711、ITU-T G.721 等。

2）原数据-IP 转换。

语音信号进行压缩编码之后，再加上包头等控制信息，形成帧以便在网络中进行传输。

3）传送。

固定电话网络中采用电路交换，在通信用户之间建立一条专用的链路进行信号传输。在 IP 网络中，将形成的语音帧放在数据报或者分组中，通过包头等控制信息，寻找下一跳信息，通过网络一跳一跳地发送到目的地。

4）IP 包-数据的转换。

IP 语音数据包到达目的地址之后，经过解压缩得到数字语音数据流，并将这一数据流提供给解码器，以便还原出原始语音信息。

5）数字语音转换为模拟语音。

将接收到的数字语音信息转换为模拟语音信息，可以还原出原始信息。总而言之，语音信号在 IP 网络上要经过模-数转换、压缩编码、IP 封装的过程，以便在 IP 网络上传，在接收端还要进行解封装、解压译码和数-模转换的过程，如图 6-16 所示。

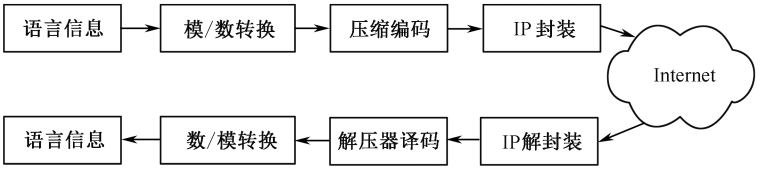


图 6-16　VoIP 传输的基本过程

（2）VoIP 电话的基本呼叫流程

1）用户摘机，VoIP 网关检测到摘机信号后，给出拨号音。

2）用户在超时之前听到拨号音后开始拨号。VoIP 网关收集用户拨打的号码，并在网络中匹配被叫用户。

3）当主叫用户网关成功匹配到被叫用户后，主叫用户通过 IP 网络发起语音呼叫，并在主叫用户和被叫用户间建立逻辑信道。

4）当被叫 VoIP 网关收到 IP 呼叫后，检测并寻找被叫用户，找到被叫用户之后并通过 PSTN 网络与之建立通话线路。

5）双方通话完毕后，任何一方挂机即结束通话。

（3）VoIP 的设备

目前 VoIP 语音通信技术已经比较成熟，并获得广泛的应用，涉及的设备包括 VoIP 网关、IP 电话交换机和 PC PBX。

1）VoIP 网关。VoIP 网关主要是完成模拟语音信号与 IP 数据包之间的相互转换，并同时和 PSTN 网络和 IP 网络连接，既可以连接传统的语音接口，又可以连接到数字电话线。VoIP 网关可以是一个专有设备，也可以简化为一个扩展卡。VoIP 网关的这种特点使得它将传统的语音通信转移到 IP 网络上，充分利用了网络资源，节省了通信成本，方便了用户的使用。

2）IP 电话交换机。IP 电话交换机是基于 IP 网络的电话交换系统，它具有传统电话交换机的功能，同时还可以提供更为丰富的接口功能，以实现语音、传真、数据、视频等业务在网络上的传送。

3）PC PBX。PC PBX 是在一台 PC 上实现 VoIP 网关和 IP 电话交换机的功能，可以进行电话交换、自动电话应答、自动呼叫分配等功能，使用灵活方便，成本较低。

2．VoIP 协议

通信 IP 网络进行语音通信，需要遵循的协议有 H.323、SIP 及 MGCP 3 种标准协议。

（1）H.323

1996 年 ITU-T 制定了 H.323 的第一版本，并在 1998 年进行了修订。H.323 协议制订的目的基于网络提供多媒体通信，为 IP 网络提供多媒体通信的基础。但是 H.323 协议并不依赖于网络架构，可以实现点对多点的组播功能，多点用户数受限，不支持群播，也不支持呼叫转移，呼叫建立时间较长。

（2）SIP

SIP 协议是由 IETF 制定的，在 OSI 属性上属于应用层协议，其可以实现语音通信、点播、

群播等功能，可以支持多媒体会议、远程教学等多媒体业务，而且具有使用灵活方便、易于扩展等特点。

（3）MGCP

H.323 和 SIP 协议是专门针对网络电话及 IP 网络所提出的两套各自独立的标准，两者间并不兼容及互通。但是 MGCP 协议与 IP 电话网络无关，主要是进行网关功能分解、掌控呼叫建立与控制。MGCP 协议可以同时支持 H.323 或 SIP 协议的网络电话系统，实现跨区域的多媒体通信。

除了上述三大协议之外，VoIP 在语音压缩编码技术方面主要有 G.729、G.723 协议，在实时传输技术方面有 RTP 传输协议，在网络传输方面，包括了 TCP、UDP、网关互连、路由选择、网络管理、安全认证及计费等相关协议。

3．VoIP 业务模式

VoIP 常见的业务模式有 H. 323IP 电话、VoBB 宽带电话和 IM 即时消息，其中 H. 323IP 电话又包括 IP 电话卡、特服号码、专线 IP 电话等，是目前国内最常见的 VoIP 语音通信模式。

6.6 接入网

随着通信新业务的不断增长，用户对用户线带宽的需求越来越高，于是在用户线概念的基础上产生了接入网的概念。接入网是指本地用户和交换机之间的数字通信技术，可以采用光纤、铜缆和无线接入等，是实现宽带化和业务综合化的关键，成为电信业的重点关注。

6.6.1 接入网简介

1．接入网的定义

接入网（Access Network，AN）是指业务节点接口（SNI）和用户网络接口（UNI）之间的具有一定传输承载能力的系统。

2．接入网的定界

如图 6-17 所示，电信网由公用电信网和用户驻地网（CPN）组成。公用电信网包括交换网（长途网）、传输网（中继网）和接入网 3 部分。核心网包含了长途网和中继网。接入网包含了核心网和用户驻地网之间的所有实施设备与线路，主要完成交叉连接、复用和传输功能，一般不包括交换功能。用户驻地网指用户终端至用户—网络接口所包含的网络部分。

接入网所覆盖的范围由 3 个接口定界，分别如图 6-18 所示。网络侧经 SNI 与业务节点（Service Node，SN）相连；用户侧经 UNI 与用户相连；管理方面则经 Q3 接口与电信管理网（Telecommunication Management Network，TMN）相连。其中业务节点 SN 是提供业务的实体，是一种可以接入各种交换型或半永久连接型电信业务的网元；而 SNI 即是 AN 和 SN 之间的接

口。可提供规定业务的 SN 可以是本地交换机、租用业务节点或特定配置情况下的点播电视和广播电视业务节点等。

（1）接入网的功能模型

接入网有 5 种主要的功能，包括用户口功能（User Port Function，UPF）、核心功能（Core Function，CF）、传输功能（Transport Function，TF）、业务口功能（Source Port Function，SPF）和系统管理功能（System Management Function，SMF）。其功能模型图如图 6-18　接入网的定界　　图 6-19 所示。

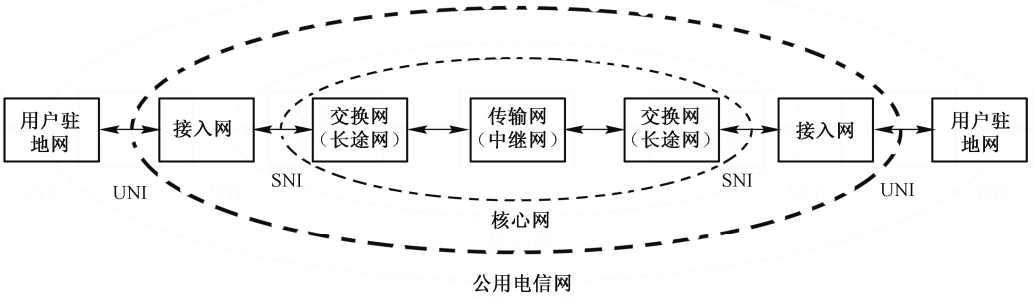


图 6-17　电信网的划分

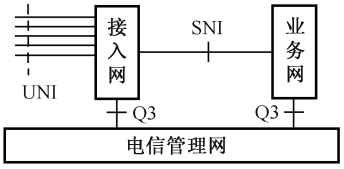


图 6-18　接入网的定界

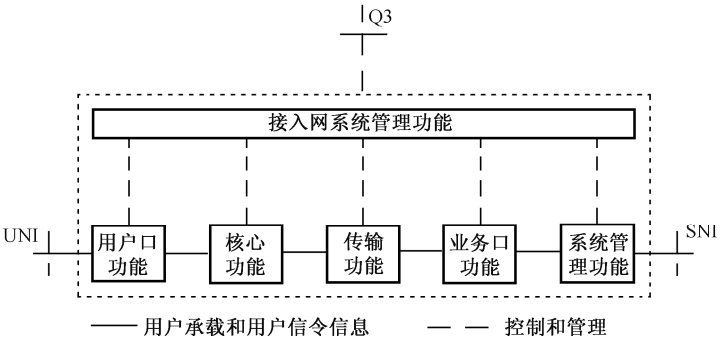


图 6-19　接入网的功能模型图

- 用户口功能：在核心功能和管理功能上融入特定的 UNI 要求，具体表现为：终结 UNI 功能、UNI 的激活/去激活、信令的转换（但不解释信令）、A/D 转换及 UNI 承载信道承载能力的处理等。
- 业务口功能：在公共承载通路上融入特定的 SNI 定义要求，以便在核心功能和系统管理功能中处理。具体功能为：终结 SNI 功能，SNI 的测试和 SPF 的维护，将承载通路的需求及操作映射进核心功能，特定 SNI 所需的协议映射等。
- 核心功能：位于用户口功能和业务口功能之间，主要用于协议处理和用于传送的复用功能。
- 传输功能：负责对接入网内分布在不同位置的公共承载体之间的传送提供通道和传输介质适配，如复用、交叉连接、物理介质、管理等功能。

- 系统管理功能：接入网系统管理功能负责对接入网中的用户接口功能、业务接口功能、核心功能和传送功能进行指配、操作和管理。

（2）接入网的特点

从图 6-17 可以看出，接入网是指交换机和用户终端之间的链路，它处于用户和核心网之间，其主要特点如下。

- 具备复用、交叉连接和传输功能。接入网主要完成复用、交叉连接和传输功能，一般不具备交换功能，它提供开放的 V5 标准接口，可实现与任何种类的交换设备的连接。
- 接入业务种类多，业务量密度低。接入网可以接入的业务种类很多，但是其业务量密度相对核心网较低。
- 网径大小不一，成本与用户有关。接入网提供交换机和用户之间的连接，网络半径相差较大。市区人口密集地区和偏远地区相差十分明显，造成用户需要分担的人均成本相差非常大。
- 线路施工难度大，设备运行环境恶劣。接入网的设备一般要在户外进行铺设，施工难度较大，而且容易受到自然和人为破坏。
- 网络拓扑结构多样，组网能力强大。接入网的拓扑结构灵活多样，可以适应各种应用，组网灵活，并且网络带有自愈功能。
- 综合性强。接入网能够承担的业务种类繁多，使用的技术也是最为繁多的。
- 直接面向用户。接入网处于网络的最边缘，直接和用户连接，当其发生问题时，用户可以直接体验到。
- 其他业务网关系最为密切。接入网是本地电信网的一部分，和本地网的其他部分关系密切。
- 快速变化、发展。接入网的技术更新较快，新技术层出不穷，促进了网络的建设和发展。
- 适应性要求较高。接入网由于要承担各种类型的业务，而且要适应各种恶劣的环境，因此接入网对网络的适应性要求很高。

（3）接入网的标准

ITU 提出的接入网标准有 G.964（V5.1 接口）、G.965（V5.2 接口）等一系列技术标准。

V5 接口是 ITU 开发的业务接点接口的一种，该接口协议结构共分为电气和物理特性的物理层 V5.1 接口、封装功能子层和数据链路子层、面向消息的协议。该接口支持公用电话网、ISDN（窄带）、帧中继、分组交换、DDN 等业务。

V5 接口具有以下 3 个优点。

- V5 接口是一个综合化的数字用户接口。
- V5 接口是一个标准化接口，可以使用不同厂商的交换机和接入设备，可以降低系统成本，优化网络，提高服务质量。
- V5 接口是一个开放的接口，交换机与设备之间连接灵活、方便，使得组网更加方便、安全、可靠。

（4）接入网的分类

根据分类的依据不同，接入网的分类方法有很多种，可以按传输介质、拓扑结构、使用

技术、接口标准、业务带宽、业务种类等进行分类。接入网目前的主要分类如表 6-1 所示。

表 6-1　接入网分类表

接入网	有线接入网	铜线接入网	数字线对增容（DPG）
			高比特率数字用户线（HDSL）
			非对称数字用户线（ADSL）
			甚高数据传输速率数字用户线（VDSL）
			单线路数字用户线（SDSL）
			速率自适应数字用户线（RADSL）

续表

		光纤接入网	光纤到路边（FITC）
			光纤到大楼（FFIB）
			光纤到家（FTTH）
		混合光纤同轴电缆接入网	
	无线接入网	固定无线接入网	微波一点多址（DRMA）
			固定蜂窝、固定无绳
			直播卫星（DBS）
			多点多路分配业务（MMDS）
			本地多点分配业务（LMDS）
			微型天线地球站（VSAT）
		移动接入网	蜂窝移动通信
			无绳通信
			卫星移动通信
			无线寻呼
			群集调度
	综合接入网	FTTC+HFC	
		有线+无线	

6.6.2　有线接入网

（1）铜线接入技术

xDSL 技术主要是在原有的电缆上采用数字信号处理技术以提高传输带宽或速率。除了话音、数据，还包括宽带的图像、视频和多媒体信息，铜线上的传输业务还涉及数字数据网（DDN）、综合业务数字网（ISDN）、会议电视、可视图文、移动通信、个人通信、计算机网等场合。上述传输技术的传输质量与光纤系统的传输质量基本相当，且成本相对较低，因而利用现有用户线快速向用户提供各种宽带业务是用户网通向未来宽带网的一座“铜桥”。

在铜线上实现高速数据传输存在群时延失真、传输频带窄、线对间串音、受用户线的物理条件及周围环境影响大和回波问题等。

在铜线上可以通过使用自适应均衡技术、调制技术、回波抵消技术等实现高速数据传输。

利用电话网铜线实现宽带传输的技术称为数字用户线技术（Digital Subscriber Line，DSL），其包括高比特率数字线、甚高速数字用户线、非对称数字用户线、单线路数字用户线等。它们的主要技术特点如表 6-2 所示。相比其他的接入方式，DSL 技术可以直接利用现有电信网络的设备开展宽带业务，而且相关设备的厂商比较支持该项技术，这些都促进了 DSL 技术的推广。

（2）非对称数字用户线

从表中可知，非对称数字用户线(Asymmetrical DSL，ADSL）在 xDSL 中性能较好，发展很快，是当前 Internet 接入网的热点技术之一，在我国各大、中城市获得广泛应用。ADSL 技术最初是为家庭用户提供点播电视（VOD）服务而开发的。Internet 的迅猛发展给 ADSL 提供了用武之地，使其发挥了巨大的作用。它最大的优点就是不需要太多改造，就可为用户提供高速的多媒体信息服务，而且不影响正常的电话或话带调制解调器通信，方便可靠，经济实用。

表 6-2　DSL 技术比较

技术方式	ADSL	HDSL	VDSL	SDSL
调制技术	DMT	2B1Q	DMT/QAM	TC-PAM
使用线对数	1 对	2 或 3 对	1 对	1 对
下行速率	1.5～8Mb/s	1.5～2Mb/s	13～52Mb/s	768kb/s
上行速率	640kb/s～1.0Mb/s	1.5～2Mb/s	1.5～2.3Mb/s	768kb/s
传输距离	5.5km 左右	3.5km 左右	0.3～1.5km	3.8～4.0km

ADSL 技术是利用现在电信网络的用户环路网的负载电话线提供非对称的，高速的多媒体信息传输，其不影响正常的电话通信，经济方便，而且随着 Internet 的发展得到了迅猛发展。

ADSL 将用户的双绞线频谱分成低频部分、上行信道和下行信道 3 部分（如**错误! 未找到引用源。**所示），低频部分用于传输语音业务；上行信道为数字信道，速率为 640kb/s～1Mb/s；下行信道为速率为 1.5～9Mb/s 的数字信道。

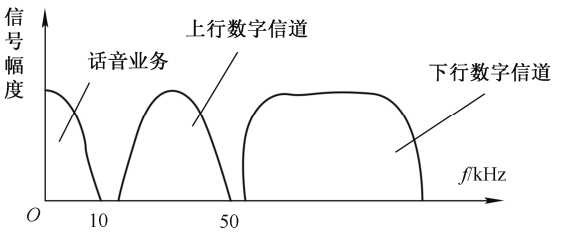


图 6-20　ADSL 系统频谱结构图

ADSL 接入网按照传输的数据格式可以分为基于分组的和基于 ATM 的接入网。ADSL 接入网的一般结构如图 6-21 所示。

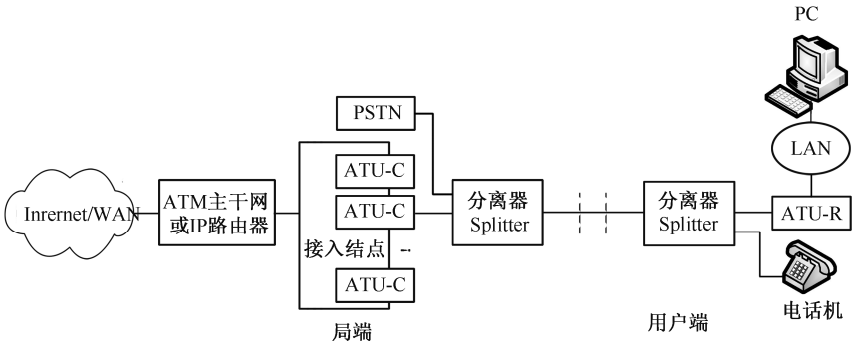


图 6-21　ADSL 接入网的一般结构示意图

ADSL 在两个方向的速率不对称，用分离器分离数据业务，使原有模拟终端继续使用，在原来电话线路只承载话音的 0~4kHz 频率上，开辟了 25~1100kHz 的数据传输频段，数据传输频段又分为上行和下行两个通道。ADSL 除了向用户提供一路普通电话业务外，还能向用户提供一个中速（速率可达 576kb/s）双工数据通信通道和一个高速（速率可达 8Mb/s）单工下行数据传送通道。

（3）高比特率数字用户线

高比特率数字用户线（High-speed DSL，HDSL）技术由于采用线对上等效频率，2B1Q 或 CAP 传输码型和回波抵消等技术，在一对普通用户线上双向传输 1.168Mb/s 的信息，在两对用户线上传输 2.048Mb/s 的信息，而且传输距离比 PCM 技术要大，因而被广泛应用于无线寻呼中继、DDN（数字数据网）、ISDN（综合业务数字网）、基站接入、帧中继、移动通信基站中继和计算机 LAN 互联网等网络中。HDSL 的通信原理如图 6-22 所示。

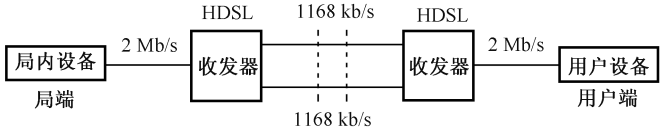


图 6-22　HDSL 的通信原理

HDSL 系统接口有 G.703、G.704、X.21 和电视会议视频接口等，其配置如**错误!未找到引用源。**所示。

HDSL 可以充分利用现有电缆来实现扩容，解决少量用户传输 2048kb/s 的宽带信号要求；通过结合语音压缩编码技术，实现线对增容；通过连接 LAN 或 WAN，传输数据及图文、影像等信息；通过与基站交换设备之间的互连，可用于移动通信；HDSL 无中继传输距离是传统的 PCM 两倍以上。但是，HDSL 不

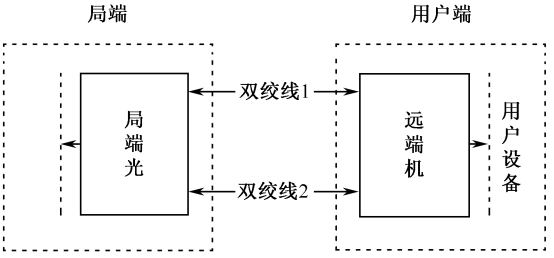


图 6-23　HDSL 配置

能传输 2048kb/s 以上的信息，传输距离限制在 6～10km 内。

（4）甚高速数字用户线

甚高速数字用户线（Very-high-speed DSL，VDSL）采用 CAP、DMT 和 DNMT 技术，上下行信道利用频分复用技术，使得最大下行速率为 51～55Mb/s，传输距离不超过 300m；当传输速率在 13Mb/s 以下时，传输距离可达到 1.5km，上行速率则为 1.6Mb/s 以上。VDSL 技术传输带宽高、速率快、成本低，可以实现宽带接入。

（5）单线路数字用户线

单线路数字用户线（Singleline DSL，SDSL)可以支持各种要求上、下行通信速率相同的应用，是对称的 DSL 技术。该技术在双线电路中运行良好，不过标准最终未确立。

（6）光纤接入技术

光纤接入网被称为光纤用户环路 FITL（Fiber In The Loop），其定义为从业务节点到用户终端之间的全部或部分采用光设备接入。根据网络单元（ONU）与用户位置的远近，光纤接入网又可分为若干种专门的传输结构，主要包括光纤到路边（Fiber To The Curb，FTTC）、光纤到大楼（Fiber To The Building，FTTB）、光纤到家（Fiber To The Home，FTTH）。

① FTTC：将光纤网络架设到小区，再通过双绞铜线或电缆接入用户，这种方式比较适合于居住密度较高的住宅区。

② FTTB：将光纤架设到办公大楼，再用电缆延伸到各用户。其适用于给一些智能化办公大楼提供高速数据、电子商务和视频会议等业务。

③ FTTH：采用光纤直接入户，不受外界干扰，无泄漏问题，供电成本低。

光纤接入网的应用类型如图 6-24 所示。

（7）混合光纤同轴电缆接入网

混合光纤同轴电缆接入网（Hybria Fiber Coaxial，HFC）是由美国贝尔实验室提出的，以模拟频分复用为基础，主干系统使用光纤，配线部分使用树状拓扑结构的同轴电缆系统传输和分配用户信息的网络，网络结构如图 6-25 所示。

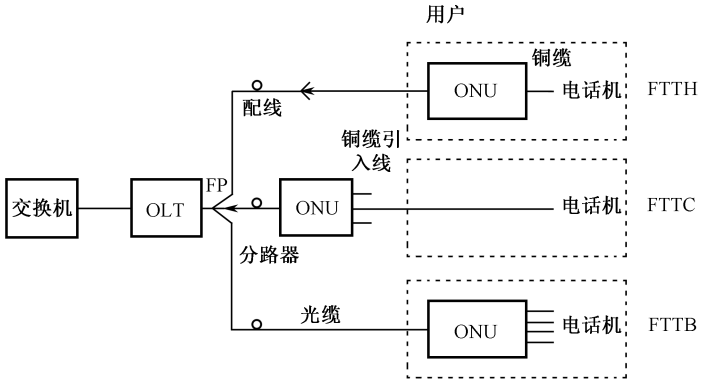


图 6-24　光纤接入网的应用类型

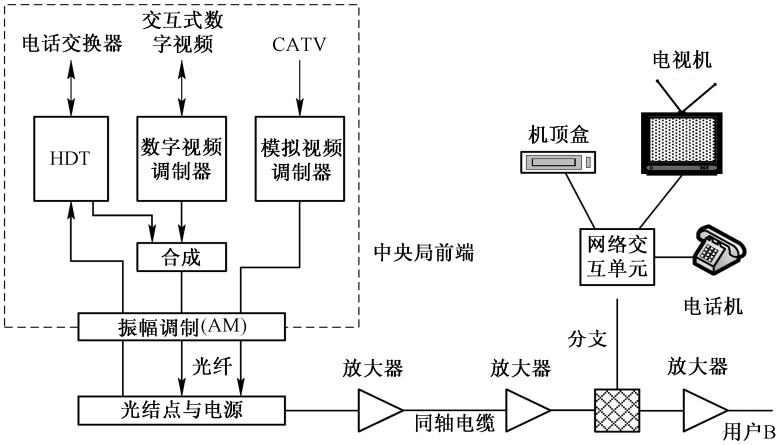


图 6-25　HFC 网络结构图

HFC 从连接技术上可以分为动态分配带宽速率和固定带宽速率两类；从传输方式划分，又可以分为对称型业务与非对称型业务两种。HFC 上行信道频率为 5～30MHz，下行信道频率为 700～750MHz。

6.6.3　无线接入技术

无线接入技术（Wireless Access Technology）又称无线接续技术，其以无线技术为媒介为用户提供多种业务的接入，使用灵活方便，已经成为目前接入的主要技术之一。

无线接入技术可分为固定无线接入和移动无线接入两种。移动无线接入系统有集群通信系统、寻呼电话系统和蜂窝移动通信系统、同步卫星移动通信系统、低轨道卫星移动通信系统和个人通信系统等。

6.7　排队论基础

排队论是通信网性能分析中的常用工具。例如，在电话网中，呼损率是衡量系统性能好坏的一个重要指标。但是网络中中继线总数往往是固定的，当用户发出局呼叫时，需要占用交换局间的中继线。为了考虑系统成本和性能的折中，需要应用排队论设计网络中应该设置多少中继线。

排队理论同样也用于解决网络延迟问题。在分组交换网中，分组信息在交换过程因为节点的存储缓冲会有延迟，延时的长短与传输链路的速率有关。由于分组交换的性能由延时的大小决定，因此在设计分组交换网时，要选择合适的缓冲器容量和链路速率，以保证分组信息的延时性能。

不仅如此，其他像局域网的介质随机访问竞争问题、ATM 网的出线竞争问题等，都需要利

用排队论来进行研究分析，因此，我们在此介绍排队论的相关基础知识。

6.7.1 排队模型基本概念

排队是日常生活中经常见到的现象，而在通信网中也常常有类似的排队现象， 通过分析这些表面现象，可以看出，排队中存在着“要求服务”和“提供服务”两方，“要求服务”即顾客，“提供服务”即服务机构。顾客要求提供服务的时间长短和到达的数目都是不确定的，这种由要求随机性服务的顾客和服务机构两方面构成的系统称为随机服务系统或排队系统。服务设施的有限性和顾客需求的随机性是产生排队现象的根本原因。排队论就是利用概率论和随机过程理论，研究排队系统内服务机构与顾客需求之间的关系，以便合理地设计和控制排队系统，使之既能节省服务机构的费用，又能满足一定服务质量要求。

排队论可以用来解决通信网中的许多问题，一个排队系统可抽象地描述如下：由于系统容量有限，顾客在系统的服务窗口前等待得到服务，窗口有空闲时顾客便立刻得到服务；若顾客长时间得不到服务，便离开窗口，或者窗口没有空闲，顾客被拒绝。排队系统的基本组成如图 6-26 所示。

图 6-26　排队系统的基本组成

排队系统由输入过程、排队过程及服务机构 3 个基本部分组成。

（1）输入过程

输入过程描述顾客按怎样的规律到达排队系统，包括以下内容。

1）顾客总体数：是指顾客的来源（简称顾客源）数量，其数目可以是有限的，也可以是无限的。

2）顾客到达方式：描述顾客到达系统是成批的还是单个的。

3）顾客流的概率分布：一般有定长分布、二项分布、泊松流（最简单流）、Erlang 分布等。

（2）排队规则

排队规则就是排队系统的类型和服务规则，其类型一般有即时制和等待制两种。其中，即时制是指服务台被占用时顾客便随即离去；等待制指服务台被占用时，顾客便排队等候服务。等待制服务规则有先到先服务、后到先服务、随机服务、有优先权的先服务、随机服务等，通常情况下是先到先服务。

（3）服务机构

服务机构可以没有服务员，也可以有一个或多个服务员。对单独顾客进行服务（即串列服务方式），即单个顾客会连续接受若干个服务员的不同服务；也可以对成批顾客进行服务，是指若干个服务员对若干个顾客提供一样的服务。

6.7.2　泊松过程

泊松过程常用于描述排队系统特性的随机过程，可以用来分析网络中的许多问题。

在介绍泊松过程之前，我们先来介绍排队系统中的 3 个基本参数。

任何排队系统都有 3 个基本参数 m、λ、μ，称为排队模型的 3 要素。

m 称为窗口数或服务员数，表征系统的资源量。它表示系统中有多少服务设施可同时向顾客提供服务，如通信系统中的中继线路数、信道数等。当 $m=1$ 时称为单窗口排队系统；当 $m>1$ 时称为多窗口排队系统。

λ是顾客到达率或系统到达率，即单位时间内到达系统的平均顾客数。λ反映了顾客到达系统的快慢程度，也反映了需要服务的一方对提供服务的一方的要求。λ越大，说明系统的负载越重。对于电话系统，λ就是单位时间内发生的平均呼叫数；对于数据传输系统，λ就是单位时间内输入的平均信息量。

μ是服务员（或窗口）的服务速率，即单位时间内由一个服务员（或窗口）进行服务所离开系统的平均顾客数。对于 $m=1$ 的单窗口系统，μ就是系统的服务速率；对于 $m>1$ 的多窗口系统，假设每个窗口的服务速率均为μ，当系统内的顾客数 k 大于 m 时系统服务率为 $m\mu$。

3 个基本参数之间的关系对排队系统的稳定性有很大影响，通常令

$$\rho=\lambda/m\mu$$

ρ 被称为排队强度，又称稳定性参数。当 $\rho<1$ 即 $\lambda<m\mu$ 时，平均到达系统的顾客数少于平均离开系统的顾客数。这时系统是稳定的，可采取即时制或等待制。当 $\rho>1$ 即 $\lambda>m\mu$ 时，平均到达系统的顾客数多于平均离开系统的顾客数，若采用等待制则可能造成阻塞，系统的稳定性无法保证。如果采用即时制，则可人为地限制系统内的顾客数量，从而保证系统的稳定性。

用如下 3 个表述对泊松过程进行定义。在时间轴上取一个很小的时隙，如**错误!未找到引用源。**所示。

1）在时隙 Δt 中有一个顾客到达的概率定义为 $\lambda\Delta t+o(\Delta t)$，$o(\Delta t)$ 表示 Δt 的更高阶项，当 $\Delta t\to 0$ 时，它更快地趋于 0；λ 是一比例常数，且 $\lambda\Delta t<<1$。

2）在 Δt 中没有顾客到达的概率是 $1-\lambda\Delta t+o(\Delta t)$。

3）到达是无记忆的，即在长度为 Δt 的一个时隙内的顾客到达与以前或以后的时隙中的到达无关。

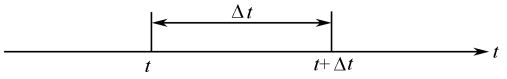


图 6-27　用于定义泊松过程的时隙

符合上述 3 点的随机过程即为泊松过程。由于在马尔柯夫过程中，事件在 $t+\Delta t$ 出现的概率只取决于在时间 t 出现的概率，因此泊松过程是马尔柯夫过程的一个特例。

利用上述 3 点，可以求得在 T 间隔内有 k 个顾客到达的概率 $p(k)$，由下式给出

$$p(k)=(\lambda T)^k m^{-\lambda T}/k!\quad (k=0,1,2,3,\cdots)$$

这就是熟知的泊松分布。其平均值 $E(k)$ 和方差 σ_k^2 由下式给出

$$E(k)=\sum_{k=0}^{\infty}kp(k)=\lambda T$$

$$\sigma_k^2 = E(k^2) - E(k)^2 = E(k) = \lambda T$$

式中，λ 为速率参数，它代表泊松到达的平均速率。

错误!未找到引用源。所示为随机到达示意图，相继到达之间的时间间隔为 τ，显然 τ 是一个连续分布的正随机变量。对于泊松到达，可以证明 τ 服从指数分布，其概率密度函数 $f(\tau)$ 可由下式给出：

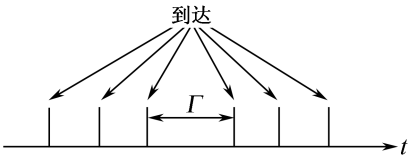


图 6-28　泊松到达的时间间隔

$$f(\tau) = \lambda e^{-\lambda\tau} \quad (\tau \geqslant 0)$$

这一指数分布的平均值 $E(\tau)$ 为

$$E(\tau) = \int_0^\infty \tau f(\tau)\,\mathrm{d}\tau = 1/\lambda$$

它的方差 σ_τ^2 为

$$\sigma_\tau^2 = 1/\lambda^2$$

值得指出的是，泊松过程有一附加特性。假定有 m 个具有任意速率 $\lambda_1, \lambda_2, \cdots, \lambda_n$ 的独立泊松流，则复合流本身也是泊松过程，其速率参数 $\lambda = \sum_{i=1}^{m} \lambda_i$。这个特性使得有多个输入端节点的性能分析得以简化。

6.7.3　M/M/1 排队模型

M/M/1 模型是一个符合泊松到达、指数服务时间、按先来先服务规则服务的单服务员排队模型。设到达率为λ，平均服务率为μ。（模型中的 M 来自马尔柯夫 Markov 过程，用来表示泊松过程或相应的指数分布。）

这是最简单的排队系统，是分析复杂排队系统的基础。该排队系统可用图 6-29 所示的排队系统模型来表示，系统的状态转移图如图 6-30 所示。

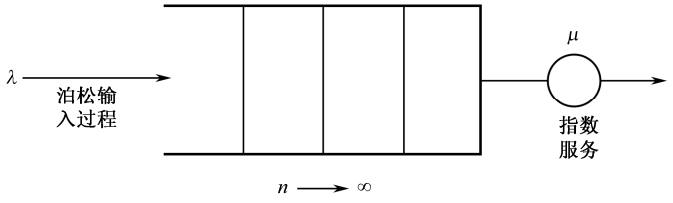


图 6-29　M/M/1 排队模型

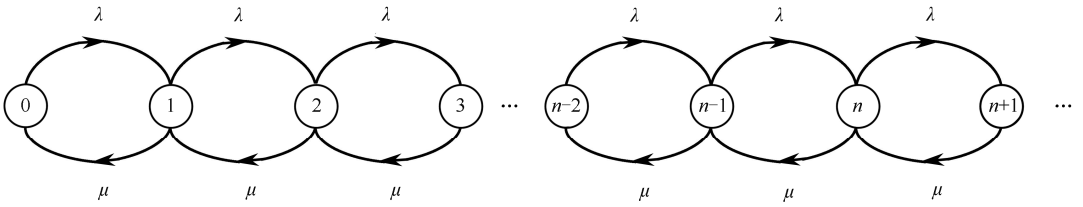


图 6-30　M/M/1 状态转移图

当系统中有 n 个顾客时，称此系统处于状态 n，与此对应出现该状态的概率为 P_n。

在系统状态图中，有顾客到达时，状态以 λ 速率向右转移一步；有顾客完成服务时，状态以速率 μ 向左移动一步。系统处于统计平衡状态时，可列出系统统计平衡方程：

$$\lambda P_0 = \mu P_1$$

$$(\lambda+\mu)P_0 = \lambda P_0 + \mu P_2$$

……

$$(\lambda+\mu)P_n = \lambda P_{n-1} + \mu P_{n+1}$$

平衡方程是通过稳态平衡原理来建立的，等式两边分别表示脱离状态 n 的速率与由状态 $n-1$ 和 $n+1$ 进入状态 n 的速率。在系统稳态平衡条件下，脱离 n 状态与进入 n 状态保持平衡，所有等式两边相等。根据此平衡方程，可得

$$P_1 = \frac{\lambda}{\mu}P_0 = \rho P_0$$

$$P_2 = (1+\rho)P_1 - \rho P_0 = \rho^2 P_0$$

……

$$P_n = \rho^2 P_0$$

当 M/M/1 排队系统的存储量为无穷大时，可以利用概率归一性条件 $\sum_0^\infty P_n = 1$ 得到 $P_0 = 1-\rho$，于是可得无限存储容量 M/M/1 排队的平衡状态概率：

$$P_n = (1-\rho)\rho^n \quad (\rho<1)$$

式中：$\rho<1$ 是上式能够成立的必要条件。为使平衡得以存在，队列的到达率或负荷必须小于输出容量 μ。如果在无限长排队模型中，P_n 这一条件不满足，队列就会随时间持续不断地增长，而永远达不到平衡点。**错误!未找到引用源。**所示为当 $\rho=0.5$ 时状态概率的图形表示。

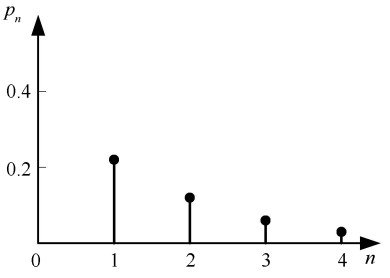


图 6-31　M/M/1 状态概率（ρ=0.5）

M/M/1 排队系统的主要性能指标如下。

1）平均队长：$E(n) = \sum_{n=0}^{\infty} nP_n = \frac{\rho}{1-\rho}$。

$E(n)$ 随 ρ 变化的关系如图 6-32 所示。

ρ	$E(n)$
0.25	1/3
0.5	1
0.75	3
0.8	4
0.9	9

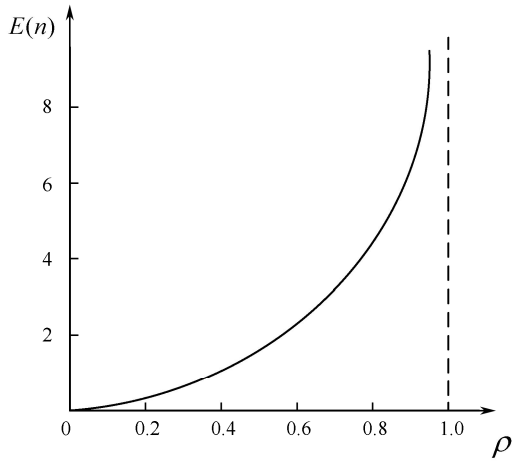


图 6-32　M/M/1 排队系统中平均队长 $E(n)$随ρ变化的关系

2）平均时延：$E(T)=\dfrac{E(n)}{\lambda}=\dfrac{1/\mu}{1-\rho}=\dfrac{1}{\mu-\lambda}$。

3）平均等待顾客数：$E(q)=\dfrac{\rho\lambda}{\mu-\lambda}$。

4）顾客平均等待时间：$E(w)=\dfrac{\rho}{\mu-\lambda}$。

5）系统效率：$\eta=\sum_{n=1}^{\infty}P_k=1-P_0=\rho$。

小结

近几年通信网发展迅速，出现了各种各样的网络。本章介绍了现有的各种通信网络技术，包括固定电话网络、X.25 分组网络、数字数据网 DDN、帧中继 FR 网、VoIP 技术、接入网和排队论基础等，希望同学能了解这些网络基本知识。

参考文献

[1]　姚军，毛昕蓉．现代通信网[M]. 北京：人民邮电出版社，2010.
[2]　石文孝．通信网理论与应用[M]. 北京：电子工业出版社，2008.
[3]　赵晓岚，胡家彦．通信网络原理[M]. 北京：国防工业出版社，2006.
[4]　夏靖波，刘振霞，张锐．通信网理论与技术[M]. 西安：西安电子科技大学出版社，2006.
[5]　张辉，等．数据通信与网络[M]. 北京：人民邮电出版社，2007.

[6]　Miller.D. 数据通信与网络[M]. 邓劝生，等译．北京：清华大学出版社，2007.
[7]　唐宝民，江凌云．通信网基础[M]. 北京：机械工业出版社，2005.
[8]　Forouzan B.A.，Fegan S.C.数据通信与网络[M]. 吴时霖，等译．北京：机械工业出版社，2002.
[9]　杨元挺．通信网基础[M]. 北京：机械工业出版社，2007.
[10]　秦国，秦亚莉，韩彬霞．现代通信网概论. 2 版[M]. 北京：人民邮电出版社，2008.
[11]　杨武军，等．现代通信网概论[M]. 西安：西安电子科技大学出版社，2004.
[12]　陆传赉．排队论. 2 版[M]. 北京：北京邮电大学出版社，2009.
[13]　何选森．随机过程与排队论[M]. 长沙：湖南大学出版社，2010.
[14]　田乃硕，徐秀丽，马占友．离散时间排队论[M]. 北京：科学出版社，2008.

第 7 章 智能网

7.1 智能网概述

随着科技的进步，通信在社会中扮演的角色越来越重要，通信技术不断演进，通信网络和设备不断更新，促进了经济的发展。21 世纪，智能网技术在全球范围内的快速发展给人们的生活带来了日新月异的变化。用户对个人通信的要求已经不仅限于普通的通话，而希望增加更多的个性化新业务，如语音邮箱、数据和图文业务等。其中，有些新业务要求网络不仅能够实现传递和交换信息，还能够完成对信息的存储、处理和灵活控制，这些新业务被称为智能业务。智能网技术深深改变了电信网提供业务的传统方式，在电信网技术发展史上是一次影响深远的变革，因为它不仅能给用户带来丰富多彩的智能业务，还为众多网络运营商和设备提供商带来了巨大利益。

智能网技术发展的目的就是促使用户方便、灵活地实现个人通信。目前，在国外已经广泛使用各种智能网业务，如“被叫集中付费业务”、“个人通信业务”、“虚拟专用网业务”等。在国家的大力扶持下，我国智能网技术也在不断发展，为适应市场需求和人们的需要，增加了网络提供新业务的能力，取得了显著的经济效益。

7.1.1 什么是智能网

智能网（Intelligent Network，IN）是 1992 年由 CCITT 制定的一个标准化名词。智能网是在原有网络通信基础上，为提供新业务而添加的一种附加网络结构。简单来说，它是以提供新业务为目的的一种技术手段，新业务的发展促进了网络的发展，使用户更加容易地控制网络，更方便地获取所需要的信息。

智能网依靠先进的 No.7 信令网和大型集中数据库支持，它将网络的交换功能与控制功能相分离，修改和增加新业务只需在业务控制点中增加和修改，并在数据库内增加业务数据和用户数据。所以，智能网是以计算机和数据库为核心的平台，或称之为智能网体系。如图 7-1 所示，

智能网体系为所有的通信网络服务，如电话网、移动通信网、IP 网等。

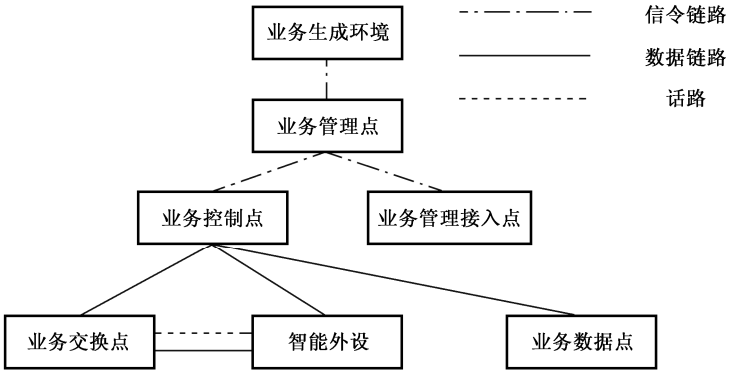


图 7-1　智能网体系的结构图

7.1.2　智能网的基本特点

1．业务控制与交换功能分离

智能网的基本思想是把传统的交换机的交换功能和业务控制功能相分离，并由附加的智能网络层完成对业务的集中控制。这样，只需要进行增加或撤销业务控制点的业务逻辑和数据的操作，即可实现新业务的增加或网上业务的撤销。同时，智能网中设置了业务生成环境（SCE）与业务管理系统（SMS），它负责生成各种新业务的逻辑和业务数据，并将新的业务逻辑和业务数据加载到业务控制点中。这样可以在网络上灵活地配置业务控制点，实现对网络的“智能化”控制。

2．业务的快速实现

智能网的体系结构支持快速生成业务。智能网技术定义的运算、比较、转移等网络功能与具体业务无关，这些功能被称为 SIB，它是智能网中组成业务的最小软件单元。智能网利用了标准的 SIB 构件业务，设计人员只要理解每个 SIB 的功能和接口的数据要求，就可以按照业务的控制需求用 SIB 构造业务逻辑。在这样的智能网体系结构下，能够快速地实现业务的生成和提供。

智能网的这种快速提供新业务的方法改变了原有电信网络的布局。同时，它是一种集最新的通信技术与计算机技术为一体的新兴技术，在实现智能网时运用了各种相关领域的新技术，如程控交换技术、信令/协议技术、大型实时数据库技术、软件技术、规范描述技术等。

3．标准化的通信接口

智能网是一个新型的网络结构，它通过消息流的交互来协调 IN 功能实体（SCP、SSP、IP 等）之间的动作，并将消息流集合成智能网应用协议（INAP）来用于 IN 功能实体之间的通信。所以，每个 IN 设备必须符合 INAP 标准的通信接口，完全通过 INAP 协议的一致性测试，才能通过国家电信机构的测试并入网，与网络上其他的 IN 设备互连互通，才是一种标准化的 IN 设备。

7.1.3 智能网和其他业务网的关系

智能网和其他业务的关系可用**错误!未找到引用源。**来表示。从**错误!未找到引用源。**中可以看出，智能网可以给现有的各种网络连通，如电话网（PSTN）、综合业务数字网（ISDN）、宽带综合业务数字网（B-ISDN）、移动通信网等，并通过这些网络向用户提供服务。

7.1.4 智能网的概念模型

智能网概念模型（Intelligent Network Conceptual Model，INCM）是设计和描述智能网体系的一个框架。它由国际电信联盟组织在 Q.1200 系列中建议提出，如图 7-3 所示。从图 7-3 中可以看出，INCM 把智能网分成 4 个层面，这 4 个平面分别为业务平面、全局功能平面、分布功能平面和物理平面，每个平面都概括地表达了由智能网所构成的网络在不同方面所提供的能力。

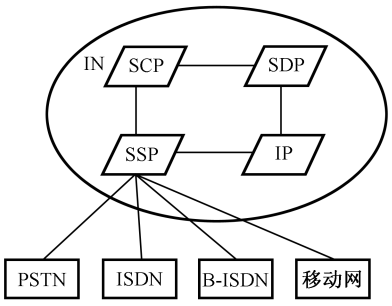


图 7-2 智能网与 PSTN、ISDN、B-ISDN、移动网的关系

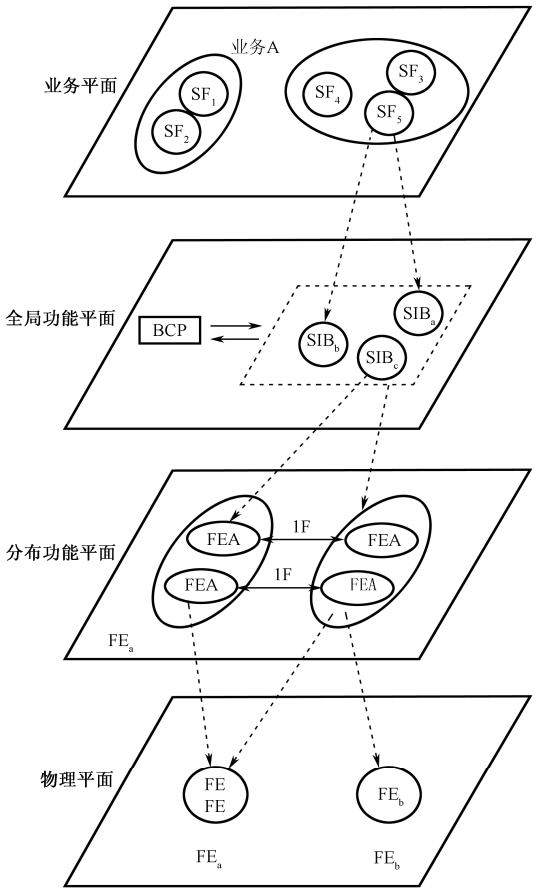


图 7-3 智能网概念模型

1．业务平面

业务平面（Service Plane，SP）描述了业务性能，与业务实现无关。它不要求业务采用什么方式来实现，只要能完成业务性能即可。ITU 在 CS-1 中共定义了 25 种具有较高商业价值的目标业务，如灵活的路由选择、灵活的计费和用户相互作用的业务，共包含 38 种业务特征。

业务属性（Service Feature，SF）是业务平面中最小的描述单位。ITU 在 CS-1 时共提出了 38 种业务属性。在 IN 的业务平面中把业务细分成业务特征的方法是一种模块化和重用化的考虑。一种业务可具有一种或几种业务属性，很多不同的业务可以有相同的业务属性。

业务平面包含对业务的制定和管理。业务制定是指一个标准业务根据具体用户的需要进行部分业务特征的修改，以构成符合用户特殊需求的业务。

在业务平面上，要考虑的一个重要问题是业务和业务特征的交互作用。因为在一次呼叫期间可能使用两种以上的业务，这时业务之间就要协同工作，业务的交互作用也就发生了，即所谓的负向业务交互。当网络中具有的业务数量不断增多以及允许为特殊用户指定业务时，业务的交互作用会越来越突出。如何消除和限制负向业务交互作用已经成为智能网理论研究的一个重要内容。

随着业务的需要，在 INCM 的第一个平面即业务平面将不断增加新的业务。到 CS-3 将会提供更多的关于宽带和未来移动对通信方面的智能网业务。

2．全局功能平面

全局功能平面（Global Functional Plane，GFP）反映了智能网实现的整体功能，把智能网看做一个整体。国际电信联盟在这个平面上定义了一些标准的可重用功能块，称为“与业务无关的构成块（Service Independent Building Block，SIB）”。SIB 是与业务无关的最小功能构件，可以完成一个独立的功能，如验证、排队等功能。在业务平面中，一个业务属性需要一个或多个 SIB 来实现。在 CS-1 中，ITU 建议了 14 个 SIB，每个 SIB 具有一种功能，目前规定的 14 个 SIB 可以满足 CS-1 中 38 种业务属性的需要。这 14 个 SIB 的名称如下。

① 算法 SIB；②计费 SIB；③比较 SIB；④分配 SIB；⑤限制 SIB；⑥排队 SIB；⑦记录呼叫信息 SIB；⑧筛选 SIB；⑨业务数据管理 SIB；⑩状态通知 SIB；⑪翻译 SIB；⑫用户作用 SIB；⑬核对 SIB；⑭基本呼叫处理 SIB。

而对于 CS-2，为了满足网间业务、多用户业务、业务管理业务和业务生成业务的要求，CS-2 增加了新的 SIB，它可以被重复使用来定义各种不同的业务和业务属性，不同的 SIB 组合方法配以适当的参数就构成了不同的业务。将 SIB 组合在一起所形成的 SIB 链接关系称为该业务的“全局业务逻辑（Global Service Logic，GSL）”。

图 7-4 给出了一个全局业务逻辑的图例。

POI（启动点）是指从交换机上报智能网业务呼叫事件，从而启动 800 号业务逻辑。POR（返回点）则是业务逻辑命令交换机根据译码结果连接主被叫，完成本次智能呼叫。BCP（基本呼叫处理）是一个特殊的 SIB，每个业务逻辑定义中都必须用到它，它负责交换机中的呼叫处理功能。

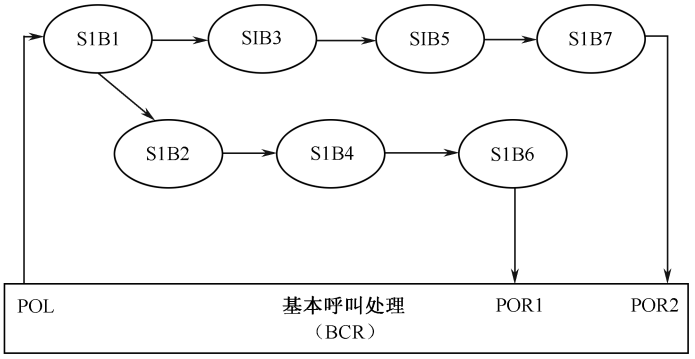


图 7-4　采用 SIB 描述方法的业务逻辑

3．分布功能平面

分布功能平面（Distributed Functional Plane，DFP）反映了智能网的各种功能模型，并说明有哪些功能实体（Function Entity，FE）和功能实体之间传送的消息，即信息流（Information Fllow，IF）。每个功能实体完成智能网的一部分特定功能，各种功能实体之间采用标准信息流进行联系。功能实体及信息流的规范描述都与它们的物理实现方式无关。

智能网的分布功能平面如**错误!未找到引用源。**所示，图中椭圆代表各功能实体，实体间的连线代表它们之间的交互关系，即信息流。

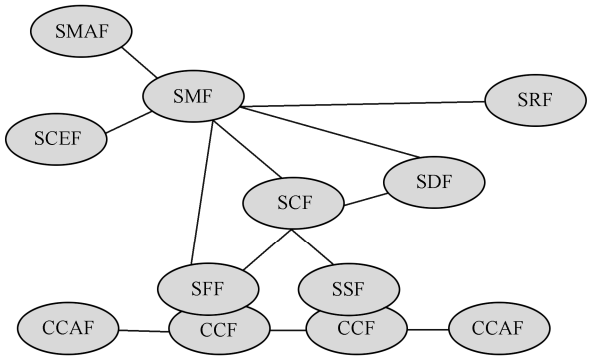


图 7-5　智能网的分布功能平面

功能平面有 9 种功能实体：CCAF、CCF、SSF、SCF、SDF、SRF、SMF、SMAF、SCEF。

呼叫控制功能（CCF）：为网络用户提供建立和控制承载业务的手段。在传统意义上它指呼叫与连接处理。

业务交换功能（SSF）：提供识别呼叫请求 IN 业务处理的手段，并与呼叫处理及这些呼叫的业务逻辑交互动作。

业务控制功能（SCF）：包括提供对呼叫请求 IN 业务进行逻辑控制的业务逻辑，并处理与业务有关的行为，如分析、翻译、筛选、寻找路由等。

专用资源功能（SRF）：通过对诸如 DTMF 接收器、语音识别功能、规程转换、通知和语音处理等资源的控制，为所有终端用户提供与 IN 网的相互联系。

业务管理功能（SMF）：提供业务准备、配置和管理控制，传送与业务逻辑和业务数据有关的信息至所有 IN 功能实体。

业务生成环境功能（SCEF）：为新的 IN 业务提供生成、证实和测试等功能，其输出包括业务逻辑和业务数据模型。

业务数据单元（SDF）：处理与业务相关的数据及网络数据的接入。

呼叫控制接入功能（CCAF）：提供至用户的业务接入，并提交给 CCF。

业务管理接入功能（SMAF）：提供至 SMF 的接口，包括审核访问功能的权限。

4．物理平面

物理平面（Physical Plane，PP）是把 DFP 中的 FE 映射到该平面的物理实体（Physical Entity，PE）上，即在物理平面上实现分布功能平面中的功能实体。一个物理实体中可以包括一到多个功能实体。但是，ITU 规定，一个功能实体只能位于一个物理实体中，不能分散在两个以上的物理实体中。

在 CS-1 阶段总共建议了 8 种物理实体：SCP（业务控制点）、SN（业务节点）、AD（附属设备）、IP（智能外设）、SSP（业务交换点）、SDP（业务数据点）、SSCP（业务交换控制点）、NAP（网络接入点）。这些物理实体具有的功能可用物理实体与功能实体间的关系来表示，如表 7.1 所示。

通过以上对智能网概念模型的 4 个平面的描述，下面概括这 4 个平面的关系。

业务平面由业务和业务属性组成，它们可以进一步采用全局功能平面中的 SIB 来加以描述和实现。全局功能平面将智能网视为一个整体，其中的每个 SIB 都完成某种标准的网络功能。每个 SIB 的功能又是通过分布功能平面上不同功能实体之间协调工作来共同完成的。不同功能实体之间的协调是通过标准的智能网接口（信息流）来实现的。以上 3 个平面之间的逻辑从上到下逐层细化。但第 3 层和第 4 层之间的关系则说明了各个功能实体是在哪些物理实体中得到实现的，是软件功能在硬件设备上的定位关系。

表 7-1　功能实体（FE）与物理实体（PE）间可能的转换

PE\FE	SCF	SSF\CCF	SDF	SRF
SCP	C	—	C	—
SN	C	C	C	C
AD	C	—	C	—
SSP	O	C	O	O
IP	—	—	—	C
SDP	—	—	C	—
SSCP	C	C	C	O
NAP	—	C（仅为 CCF）	—	—

注：C 表示必要的功能；O 表示可任选的功能；—表示不允许的功能。

7.2　智能网的结构

智能网是由一些功能实体组成的网络部件构成的，智能网的构成有两种含义：一个是这些功能实体的内部功能结构，另一个是由这些功能实体所构成的网络结构。具体来说，智能网

一般是由业务交换点（SSP）、业务控制点（SCP）、业务数据点（SDP）、信令转接点（STP）、智能外设（IP）、业务管理系统（SMS）、业务生成环境（SCE）等几部分组成的。

7.2.1 业务交换点

业务交换点（Service Switching Point，SSP）是 PSTN、移动网与智能网的连接点，主要功能是提供接入智能网功能集，其可检出智能业务的请求，并与 SCP 通信；同时可以对 SCP 的请求做出响应，允许 SCP 中的业务逻辑影响呼叫处理。从功能上讲，一个业务交换点包括呼叫控制功能（Call ControlFunction，CCF）和业务交换功能（Service Switching Function，SSF）。在没有建设独立的 IP（智能外设）的情况下，SSP 还应包括部分的专用资源功能（Specialized Resource Function，SRF）。呼叫处理功能能够完成客户呼叫、呼叫建立和呼叫保持等基本接续功能。业务交换功能是指接收和识别智能业务呼叫，并向业务控制点报告，同时接收业务控制点发送的控制命令。

（1）功能

在 SSP 中实现 CCF 和 SSF 的功能，其主要功能如下。

1）全部呼叫控制功能。

2）识别 IN 呼叫，向 SCP 报告，并准备向 SEP 报告该检测点，请求接续控制。

3）区分检测点类型（TDP-R、TDP-N、EDP-R、EDP-N）并做出相关控制处理。

4）根据 SCP 的命令继续或改变呼叫处理流程。

5）维护与 SCP 的通信接口。

（2）SSF 模型

SSF 包括基本呼叫管理（BCM）、智能网交换管理（IN-SM）、属性交互作用/呼叫管理（FIM/CM）及 SCF 的通信管理几部分。

（3）关键技术

实现 SSP 系统的关键技术如下。

1）在原有程控交换机软件中建立并实现 SSF/CCF 模型。

2）实现 INAP 通信协议（SSP-SCP、SSP-IP）。

3）实现 7 号命令的 TCAP 接口。

4）实现必要的语音提示和双音多频接收器资源提供，以支持 SRF 功能。

7.2.2 业务控制点

业务控制点（Service Control Point，SCP）是智能网的核心构件，管理智能网所有业务的控制功能，具体功能是接收 SSP 送来的查询信息，查询数据库，并进行各种译码；同时，SCP 能根据 SSP 上报的呼叫事件启动不同的业务逻辑，然后向相应的 SSP 发出呼叫控制指令，从而实现各种智能呼叫。

SCP 为了保证服务不间断，在实际配置中都是采用双备份的，一般由小型机、高性能微机

和大型实时高速数据库组成。

（1）功能

1）SCF 功能：

① 对智能业务呼叫进行控制和处理。

② 维护与 SSF、SRF、SDF 的通信接口，对消息做协议转换处理。

③ 接收 SMP 的管理，完成 SMP 的统计和管理指令，并报告结果。

2）SDF 功能：

① 对实时数据库进行统一的数据访问和安全管理。

② 与 SCF 进行通信，交互访问数据库的命令和数据。

③ 与 SMP 进行通信，接收 SMP 下载的系统数据和业务数据。

（2）SCF 和 SDF 的模型

SCP 分为业务逻辑执行环境（SLEE）和业务逻辑程序库（SLPLibrary）两部分。SLEE 可以调用一个或多个业务逻辑处理程序在其中运行，完成相应的业务逻辑处理。SLEE 包括如下部分。

1）业务逻辑执行管理（SLEM）。

2）SCF 数据存取管理。

3）功能选取管理。

4）功能实体通信管理（FEAM）。

5）业务逻辑程序管理。

各部分的主要功能介绍如下。

1）业务逻辑执行管理：由业务逻辑处理实例（SLPIs）、业务逻辑选择/交互管理（SLSIM）和资源管理 3 部分组成。

SLEM 处理并控制总的业务逻辑执行动作。SLEM 对业务逻辑执行的控制一般是通过有限状态自动机来实现的。SCP 对每一个 IN 呼叫都要调用相应的业务逻辑程序来执行，这个执行的过程就是一个业务逻辑处理实例。

2）SCF 数据存取管理：该模块提供 SCF 中共享和固有信息（在 SLPI 生存期之外的信息）的访问、存储和管理功能，访问远端 SDF 的功能。这些功能是在与 SLEM 模块进行通信的过程中提供的。

3）功能选取管理：该模块用于通过 FEAM 接收和分配选路功能库中的功能程序。它也管理一个特定功能选取的增加、删除和挂起。

4）功能实体通信管理（FEAM）：该模块提供 SLEM 与其他功能实体通过消息进行通信的功能。

5）SLP 管理：该模块管理其他实体与业务逻辑处理程序的选定和分配。它和 FEAM 协同工作，也对一个待定 SLP 的增加、删除和挂起进行管理，如接收来自 SMP 的 SLP。

（3）关键技术

实现 SCP 系统的关键技术如下。

1）根据智能网标准体系结构原理实现 SCP 系统。

2）实现符合国际 INAP 的通信协议。

3）实现新业务动态加载的机制。

4）大容量实时数据库技术。

5）支持高处理能力及容错性能的软件设计。

7.2.3 智能外设

智能外设（Intelligent Peripheral，IP）的任务是协助完成智能业务的特殊资源，它可以是一个独立的物理设备，也可以是 SSP 的一部分，接受 SCP 的控制，执行 SCP 业务逻辑所指定的操作。也可以在网络中集中设立 IP，其功能与其他交换机共享，可以节省成本，便于语音资源的统一管理和开展。

智能外设的实现有两种方式，一种是综合 SSP 方式，一种是独立 IP 方式。

随着智能业务的不断发展，智能业务对专用资源功能的要求越来越多样化。不仅要求提供更丰富的专用资源设备，还要求其控制能力从资源管理领域扩展到系统与用户交互的管理领域，所以需要采用独立智能外设的方式。

（1）功能

1）具有对资源进行控制、分配并管理资源状态、时隙交换功能。

2）维护与其他功能实体的通信接口。IP-SCP 之间使用 INAP 协议完成通信，IP-SSP 之间完成话路的承载连接。

3）IP-SMS 之间完成维护、管理命令及数据的传送。

4）在 CS-2 建议中，SRF 的控制功能增加了对用户交互描述的控制能力和多媒体数据的存储管理能力。

5）对资源、业务、呼叫、操作的统计功能。

（2）SRF 的功能模型

SRF 包括特殊资源管理和功能实体通信管理。

1）特殊资源的管理：提供对 SRF 内部资源的管理，包括对一个资源的占用和释放，管理资源的状态，控制资源的动作。

2）功能实体通信管理：该模块提供与其他功能实体通过消息进行通信的功能。SRF 与实体 SSF/CCF、SCF 和 SMF 有通信联系。

（3）关键技术

实现 IP 技术，特别是独立 IP 系统的关键技术如下。

1）建立并实现 SRF 模型。当具有用户交互描述机制时，要在 IP 软件中设计实现 UIScript 的执行环境，并且支持对 UIScript 的下载和执行。

2）实现 INAP 的通信协议。

3）实现 TCAP 和 ISUP 接口。

4）大型实时数据库技术。

5）大容量、高处理能力的容错平台的实现。

7.2.4　业务管理点

业务管理点（Service Management System，SMS）一般具备 5 种功能，即业务逻辑管理、业务数据管理、网络配置管理等。SMS 具体由业务管理点（Service Management Point，SMP）和业务管理接入点（Service Manage Access Point，SMAP）组成。SMP 为服务器端，SMAP 为客户端。SMAP 提供了访问 SMP 的界面，在 SMAP 操作的结果都存放在 SMP 上。

1）SMS 是智能网的管理中心，其管理功能如下。

① 业务部署功能。

② 业务提供功能。

③ 业务运行控制功能。

④ 账务功能。

⑤ 业务监视功能。

⑥ 网络管理功能。

⑦ 接入管理功能。

2）ITU 在 CS-1 建议中没有定义 SMS 的功能模型，但 SMS 的实现可以参考电信网络管理（TMN）的原理进行。

TMN 为电信网和业务的管理提供了性能管理、故障管理、配置管理、计费管理和安全管理这 5 种管理功能，并把这些管理功能分为事务管理层、业务管理层、网络管理层和网元管理层 4 个层次，这 4 个层次分别代表了最高管理者、业务运营者、网络的组织管理者及一般设备的操作维护人员对电信网管理工作的不同要求。

3）实现 SMS 的关键技术如下。

① 遵循 TMN 思想，结合 IN 管理要求构造 SMS。

② 分布处理系统技术。

③ 高处理能力及容错性能的软件设计。

④ 独立于业务的用户接入系统设计。

7.2.5　业务生成环境

业务生成环境（Service Creation Environment，SCE）的功能是根据客户的需求生成新的业务逻辑，其提供了友好的图形编辑界面和标准图元，为新业务提供设计开发、验证和仿真调试、加载功能。设计好的业务需要通过严格的验证和模拟测试，以保证其不会给电信网已有业务造成不良影响。

（1）SCE 的功能

根据业务实现的要求，SCE 应该具有如下的功能。

1）业务设计功能：

① 根据 SIB 的功能生成业务属性逻辑或者业务逻辑。

② 根据 SEB 中 SSD、CID 的格式生成业务数据。

③ 由业务逻辑和业务数据生成业务。

④ 对业务、业务逻辑和业务数据进行管理，包括增加、删除和阅览等功能。

2）业务的验证和模拟功能：对生成的业务逻辑做合法性检验，并在模拟环境下进行该业务运行的模拟，看结果是否符合业务的描述，如果与预计设计不相符，则需要对设计业务逻辑做修正。

3）业务的装载功能：将生成的业务逻辑和业务数据传送给 SMF，由 SMF 向 SCF 加载。

（2）业务生命周期

业务生命周期描述了业务在生存期内经历的所有阶段，包括需求分析、业务生成、业务测试、商业运营 4 个阶段。业务生成阶段包括业务设计、业务编辑、业务仿真/调试 3 个过程。业务测试是验证所生成的新业务是否完整的有效手段。

（3）关键技术

实现 SCE 系统的关键技术如下。

1）IN 业务的形式化描述方法。

2）SIB 及软件重用理论的研究。

3）业务验证的理论与方法。

4）业务属性交互作用的机理及处理方法。

5）业务生成管理的方法，支持将生成的业务逻辑和数据向 SMS 加载。

7.3 移动网与智能网的结合

目前，移动网发展迅猛，已经从第 2 代移动通信系统发展到第 4 代移动通信系统，而且已经在世界范围内得到普及，可以提供应急救援通信、航空服务、海上服务和陆地移动通信等业务。智能网和移动网相结合，可以更加经济、方便地提供新业务。

7.3.1 CAMEL 结构、协议及业务

移动网络增强型逻辑的客户化应用（Customized Application for Mobile Network Enhanced Logic，CAMEL）是由 ETSI 于 1997 年首次推出的，目的是在 GSM 中引入一个智能网平台以便设计新的增值业务，这些增值业务可以不用漫游就享受与归属网络同样的服务。CAMEL 标准目前可以分为 4 个阶段，分别为 CAMEL1 阶段、CAMEL2 阶段、CAMEL3 阶段、CAMEL4 阶段。

其中，CAMEL2 支持的属性如下。

1）移动发起和前转的呼叫。

2）移动终止的呼叫。

3）随时的查询。

4）消除录音通知。

5）录音通知和带内信息交流。

6）计费信息。

7）补充业务调用通知。

7.3.2　CAMEL 网络结构

图 7-6 所示为 CAMEL1 的网络体系结构，从图中可以看出，CAMEL1 的网络体系结构实体既含有 GSM 交换网原有的实体，如归属位置寄存器（HLR）、拜访位置寄存器（VLR）、移动交换中心（MSC）与网关移动交换中心（GMSC），也有为了实现 CAMEL 业务而新引进的智能网的实体，如 GSM SSF（GSM 业务交换功能）与 GSM SCF（GSM 业务控制功能）。实体之间除了原有通信协议又增加了一些新的协议，如在 HLR 协议上添加了移动发送端的 CAMEL 业务信息（O-CSI）和移动终端的 CAMEL 业务信息（T-CSI），以及补充业务 CAMEL 的业务信息（SS-CSI）。

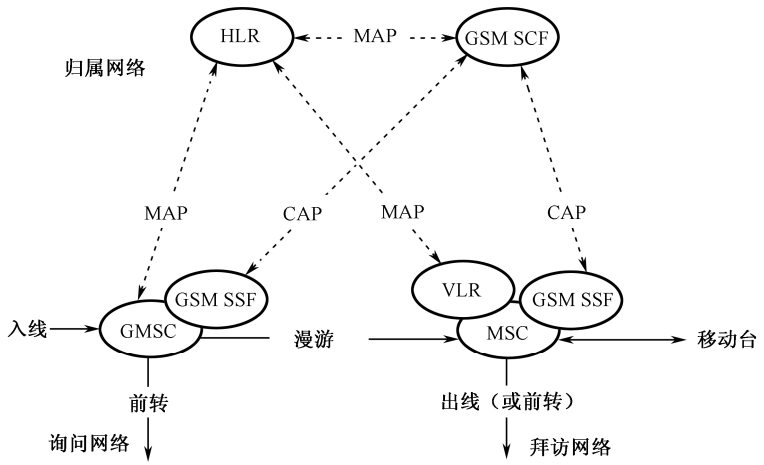


图 7-6　CAMEL1 的网络功能结构

HLR——归属位置寄存器；MSC——移动交换中心；SSF——业务交换功能；
VLR——拜访位置寄存器；MAP——移动应用部分；SCF——业务控制功能；
GMSC——入口移动交换机；CAP——CAMEL 应用部分；SRF——专用资源功能

7.3.3　CAMEL 协议

CAMEL 体系结构中主要用到了 CAP 协议和 MAP 协议，下面分别对它们做简单介绍。

（1）CAP 协议

CAP 协议主要应用在 GSM SSF、GSM SCF 和 GSM SRF 之间。CAP 主要是基于 ETSI 的 INAP 制定的。INAP 共有 29 条操作，CAMEL 仅涉及其中的几条，且操作的名称相同，但参数

有所不同。

在 CAMEL1 中，没有 GSM SRF 功能实体，仅有功能实体 GSM SSF 与 GSM SCF，所用的 CAP 协议有 7 条操作，分别如下：①ActivityTest；②Connect；③ Continue；④EventReportBCSM；⑤InitialDP；⑥ReleaseCall；⑦RequestReportBCSMEvent。

在 CAMEL2 中，增加了 gsmSRF 功能实体，所以又增加了 15 条操作，分别如下：ApplyCharging， ApplyChargingReport， AssistRequestInstructions， CallInformationReport，CallInformationRequest，Cancel，ConnectToResource，DisconnectForwardConnection，Establish TemporaryConnection， FurnishChargingInformation， PlayAnnouncement， PromptAndCollect UserInformation，ResetTimer，SendChargingInformation，SpecializedResourceReport

（2）MAP 协议

为了实现移动网与智能网的互连，提供 CAMEL 业务，需要将原有的 MAP 协议升级为 MAP Phase 2+协议。MAP Phase 2+协议为了实现 CAMEL 业务而对原有的 MAP Phase 2 协议做了一些修改，增加了 HLR 与 GSM SCF，GSM SCF 与 MSC 之间的接口，并在原有的内容中增加了涉及 CAMEL 签约信息参数等内容。

其中，GSM SCF 与 HLR 之间增加的信息流主要为了支持 GSM SCF 在任意时间向 HLR 询问用户信息的请求，以及 HLR 对此请求的响应。网络运营商可以安排 HLR 接收 GSM SCF 的请求或者拒绝其请求；新增加的 MSC 与 GSM SCF 之间的信息流只有一条，即补充业务通用通知，主要用于 MSC 向 GSM SCF 通知所调用的补充业务。在 HLR 与 VLR 之间的信息流主要有：删除用户数据请求与响应，插入用户数据请求与响应。为了支持 CAMEL 业务，在这些信息中增加了有关 CAMEL 业务数据的参数项。

7.4 宽带智能网

随着经济和社会的发展，人们对业务的需求越来越高，传统的语音业务已经不能满足人们日益增长的需求，人们希望有更多的多媒体业务。宽带智能网是智能网与宽带综合业务数字网的综合，它提供了一个可编程的业务平台，能够实现新业务的灵活加载，快速开发出声音、图像、文字等数据多媒体业务，从而适应不断增长和变化的客户化需求。

7.4.1 智能网与宽带综合业务数字网的互连

建立宽带通信系统的目的是提供符合市场需求的多媒体业务，典型的多媒体业务有宽带视频会议、视频点播（VoD）、TV 分配和宽带虚拟专用网（B-VPN）等。当前，多媒体业务的开发基本上采用了不同的体系结构和实现方案。因此，随着新业务的不断出现，每一种新业务都要涉及一套新设备，开发新业务的投资越来越大，合理利用资源的问题也变得愈发突出。

ITU 从 20 世纪 90 年代初就着手研究智能网与宽带综合业务数字网结合的问题，提出了 IN CS-4 等一系列建议草案。

宽带智能网的主要特点如下。

1）与业务有关的控制功能和与业务无关的控制功能分离。

2）业务控制功能与网络资源管理功能分离：智能网与宽带综合业务数字网的综合要求在同一平台上实现这些多媒体业务，而不是为每一种业务分别配一套专用设备。

3）呼叫控制与承载控制分离，提供真正的多点连接的宽带多媒体业务：它的呼叫模型将考虑宽带综合业务数字网的呼叫与承载连接分开的概念，还要进一步研究呼叫方处理（CPH），以支持多方控制，真正做到与宽带综合业务数字网的综合。

智能网与宽带综合业务数字网的综合基于以下关键技术。

1）宽带智能网允许多个业务逻辑同时作用于一个呼叫，即由多个 SCF 同时处理一个呼叫，这称为多点控制。多点控制能力为实现复杂的多方多媒体业务提供了基础。

2）宽带智能网的流量控制功能。

3）宽带智能网中的信令系统。

4）提供宽带智能业务中所必需的专用资源，以及这些专用资源与宽带智能网中功能实体之间的信令。

5）宽带智能网互连。

6）宽带智能网的安全问题。

7）确定宽带智能网的体系结构。

8）建立宽带智能网的所有功能实体结构。

7.4.2　宽带智能网的体系结构与呼叫模型

1．体系结构模型

在分析和研究宽带综合业务数字网中的多媒体业务的特点后，ITU 设计了智能网与宽带综合业务数字网结合的参考体系结构模型，如图 7-7 所示，图中显示了智能网和宽带综合业务数字网中的主要物理实体和功能实体，以及它们之间的信令管理。除本地交换机（LEX）外，每一个物理实体只用一个实例来表示。在该参考体系结构中，还显示了将智能网和宽带综合业务数字网的功能实体映射到物理实体的方案。功能实体之间的接口可以采用宽带智能网应用协议（B-INAP）、宽带综合业务数字网信令（UNI、NNI），而功能实体内的接口则采用内部软件接口。为了方便描述和理解，本模型将智能网概念模型中物理平面和功能平面结合在一起考虑。

在宽带智能网的体系结构中，宽带综合业务数字网的功能实体包括：连接控制（Link Control，LC）、终端控制（enD Control，DC）、边控制（Edge Control，EC）。与宽带综合业务数字网的功能实体相对应的用户侧的功能实体包括：连接控制（Link Control Agent，LCA）、终端控制（enD Control Agent，DCA）。

2．呼叫状态模型

基本呼叫管理（BCM）模型是在智能网和 ATM 交换机之间引入的模型，它可以灵活、有效地控制和管理网络资源，使宽带智能网对呼叫事件有比较好的可视性，智能网根据 BCM 内

的标识状态来“观察”智能呼叫的全过程。

ATM 交换机的功能实体是 B-CCF 实体，由呼叫控制、承载连接控制以及 BCM 3 部分共同构成。BCM 并不是一个功能实体，它的功能分布在 B-CCF 和 B-SSF 之间。它是对 ATM 交换机实现用户之间通信链路的基本呼叫控制和承载连接控制过程的抽象，ATM 交换机无法处理的呼叫控制和承载连接控制事件都能被 BCM 检测到，然后上报 B-SSF，并触发智能网业务逻辑。

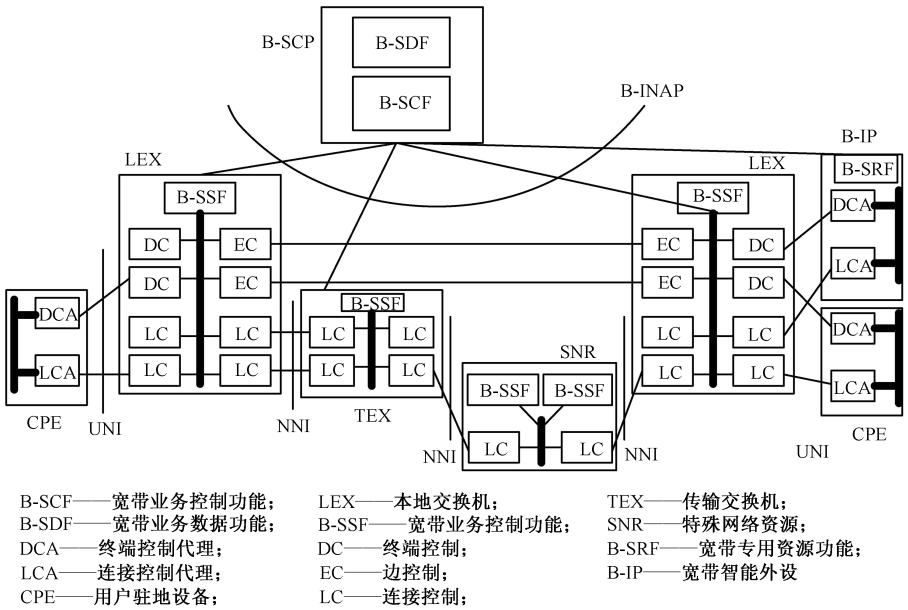


图 7-7　智能网与宽带结合业务数字网结合的参考体系结构模型

基本呼叫状态模型（BCSM）是 BCM 的核心部分，由呼叫点（PIC）、检测点（DP）、转移 3 个基本组件组成，它用高级的有限状态自动机对交换机的基本呼叫处理进行模型化，描述了 B-CCF 为建立和维护用户通信链路所要求的动作，标识基本呼叫处理的各逻辑点。

1）智能网业务逻辑请求有关的基本呼叫和连接操作后的状态都是由 PIC 标识完成的。

2）当 DP 在 PIC 上检测到某些特定的事件时，即触发智能网业务处理该动作。BCSM 在该 DP 上等待 B-SCF 发回指示，如果 B-SCF 判断触发条件满足，则按业务逻辑命令 BCSM 进行相应处理，否则，BCSM 继续下面的呼叫处理，在 DP 上可以转移控制。

3）基本呼叫处理从一个 PIC 到另一个 PIC 的正常流向称为转移。

在宽带综合业务数字网中，呼叫控制和承载控制需要信令中分离，即首先建立呼叫连接，然后建立承载连接。因此，宽带智能网标准在交换网中定义了两组基本呼叫状态模型，一是用来表示呼叫过程中主叫方的呼叫状态的发方 BCSM，BCSM 又分为呼叫控制 BCSM 和承载控制 BCSM；二是用来表示呼叫过程中被叫方的呼叫状态的收方 BCSM。

7.5　智能网与因特网互连

当今世界，Internet 以 TCP/IP 通信协议连接各个国家和各个部门计算机网络的数据通信网，在 Internet 上集中了各个部门、各个领域、各种信息资源的共享数据库，形成全球最丰富、大量的信息源。通过光纤电缆与 Internet 网互连，用户可以进行跨国界通信交流，查询网上的数据库获取信息，共享远端计算机的资源。

PSTN 业务和 Internet 业务的本质目的都是更好地实现人与人之间的通信与交流。在用户看来，Internet 和 PSTN 发起或接收一次呼叫的目的都是一样的，如果把两种呼叫方式以某种形式统一起来将会使用户更方便地使用。所以，智能网发展的热点之一是与 Internet 结合，实现 PSTN 与 Internet 的互通，提供新业务。其主要的代表业务如下。

1）点击拨号：当用户在计算机上访问 Internet 时，可以单击某个按钮，发起一个 PSTN 呼叫；同样可以发送/接收传真。

2）通知来话：当用户在计算机上访问 Internet 时，有来自 PASN 对该用户的呼叫，可以在屏幕上显示出来，以便用户决定如何处理该呼叫。

3）Internet 电话：Internet 用户可以利用计算机直接与 PSTN 的普通用户进行通话。

7.5.1　IN 与 Internet 的互连方案

1．IETE 提出的互连方案

PINT 将业务分为业务请求和业务执行两个阶段。业务请求通过 Internet 进行非话音的交互操作来完成，而语音和传真等信息则仍由 PSTN 网承载。PINT 业务如下。

1）点击拨号。

2）点击发送传真。

3）点击回送传真。

4）语音接入信息内容。

图 7-8 所示是为实现 PINT 提出的 4 个业务而设计的接口体系结构。

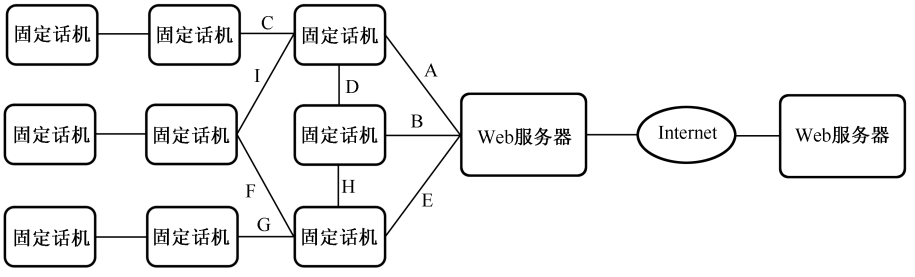


图 7-8　PINT 接口体系结构

① A 接口传送从 Internet 到业务节点的业务请求。

② B 接口传送从 Internet 到业务管理系统的业务管理请求。

③ C 接口传送从业务节点到交换机的呼叫控制请求。
④ D 接口是业务管理系统对业务节点执行管理操作的接口。
⑤ E 接口传送从 Internet 到业务控制点的业务请求。
⑥ F 接口传送从业务控制点到移动交换中心的业务呼叫控制请求。
⑦ G 接口传送从业务控制点到业务交换点的业务控制请求。
⑧ H 接口是业务管理系统对业务控制点中心管理操作的接口。
⑨ I 接口传送从业务节点到移动交换中心的业务呼叫控制请求。

2．ITU 提出的互连方案

IN 与 Internet 的互连方案最早是由 ITU 在 IN CS2 提出的。在该方案中 IN 通过新增的互连代理功能（Interworking Agent Function，IAF）实现与 Internet 的互连。IAF 的主要功能有接收 SCF 指令并将其转换为 HTML 发给 Internet 用户；接收用户通过 HTTP 发来的请求并转换成对 SCP 的指令；从 Web 服务器接收命令转换成 SCF 的指令；维护 IN 用户接入 Internet 时 PSTN、ISDN 连接的逻辑关系。

目前 ITU 的新方案已确定了支持网络互连的增强的 IN 分布功能平面结构，并定义了 IN 与 Internet 在业务管理层、业务控制层和传输层 3 个层面上的互连关系，其总体结构如图 7-9 所示。

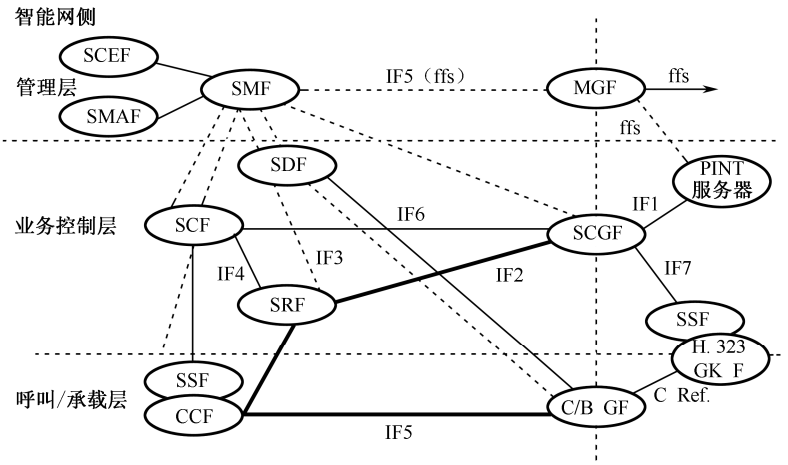


图 7-9　IN 与 Internet 的互连功能结构

在 ITU 的互连方案中，有 3 个层次都增设了用于网间互连的网关实体，包括管理网关（MGF）、业务控制网关（SCGF）、呼叫承载网关（C/B GF），这些网关功能可以合并到 IN 或 Internet 的物理实体中，也可以是独立存在的实体。图 7-9 中 IF 表示信息流接口，ffs 表示尚未确定。

（1）业务管理层互连

业务管理方面的交互问题主要靠业务管理层的互连解决。在 IN 侧，设计的功能实体有 SMF、SMAF 和 SCEF，ITU 建议这几个功能实体通过新增的功能实体 MGF 和 Internet 交互。

（2）业务控制层互连

在业务控制层上，不仅要增加功能实体 SCGF，还需要对 IN 侧的 SCF、SDF、SRF 及 Internet 侧的 H.323 网关进行相应的功能扩充。在涉及实时的业务控制时，通过 SCGF 来交互 Internet

与 IN 的信息，一方面，能发现并激活发生在 Internet 上的互连业务，将业务的请求上报给 SCF；另一方面，SCF 能将处理的结果返回给 Internet。

（3）传输层互连

随着 IP 电话网关和 GSTN 接口的标准化， H.323 定义的 Internet 多媒体会议可以纳入到这个体系结构中，为支持传输层的互连，也需要增加一个新的功能实体（C/B GF）。C/B GF 的基本功能包括电话网语音到 Internet 语音编码的转换功能、IN 网侧 CCF 与 Internet 之间承载信息的互通功能和具有支持拨号接入 Internet 的功能。

（4）IN/Internet 互连结构的相关协议

由于 ITU 的 IN/Internet 互通结构中定义了新的功能实体，并对一些原有功能实体的功能进行了扩充，因此需要定义新的接口及消息流，并对已有的进行扩充。下面对图 7-9 中标注的一些接口进行简单的讨论。

① IF1：PINT 服务器-SCGF 接口，可采用 PINT 协议。

② IF2：SCGF-SRF 接口，用于实现需要从 IP 网向 IN 侧传递目标业务数据的业务。

③ IF3：SCGF-SCF 接口，需能反映与 IF1 相关的需求。可采用类似于 INAP 的协议，或对 INAP 协议进行扩充。

④ IF4：SCF-SRF 接口，需要对现有的 INAP 标准进行增强。

⑤ IF5：CCF-C/B GF 接口，承载连接建立信令。

⑥ IF7：H.323 GKF（软 SSF）-SCGF 接口，用于触发和控制从 Internet 侧的 H.323 GKF 发起的增值业务，可采用现有的 SCF-SSF 的 INAP 协议。

与此同时，IP 的管理终端与 MGF 的接口可采用 HTTP，而 MGF 与 SMS 系统的接口采用 SNMP，或在已有的网络通信协议中嵌入所需的管理类消息乃至自定义新的互连协议。

7.5.2　IN/Internet 互连的安全问题

GSTN/IN 是一个相对独立的、用户终端功能有限的系统，安全问题不是十分突出。但是 Internet 的所有设备都处在开放的环境下，而且用户终端功能非常强大，所以 IN/Internet 互连面临着严重的安全问题。智能网把 PSTN 和 Internet 连接起来，就很可能把电信网的核心部分暴露给 Internet，导致出现安全问题，影响整个电信网的安全。

从用户的角度来看，安全问题主要有用户的恶意攻击和无意失误。Internet 中主要考虑一些用户的恶意攻击，不考虑用户的无意失误。但是，用户的无意错误对 IN/Internet 综合业务的影响可能会比较大，因此，在 IN/Internet 互连结构中对用户的恶意攻击和无意失误都需要进行深入研究分析。

7.6　智能网与电信管理网

ITU 开发了对电信网实行统一的综合维护管理的新手段——电信管理网（Telecommunications Management Network，TMN），目的是适应电信网技术的飞速发展和各种通信新业务的需要。

TMN 提供了一个全球都接受的框架，从而实现对各种类型的通信业务、网络及网络元素的统一管理，使世界变成地球村。同时，它使得网络管理系统与电信网在标准的体系结构下，按照标准的接口和标准的信息格式交换管理信息，从而实现集中的、全面的自动化网络管理功能。

7.6.1 TMN 的产生和发展

在电信网络迅速发展、日新月异的今天，新业务层出不穷，电信网络设备种类繁多。许多电信设备厂商在电信市场上的激烈竞争，导致电信网络设备状况日趋复杂。许多网络都有各自的网络管理系统，有着各自不同的体系结构，使用不同的管理接口和管理操作，管理特定的网络单元和设备。在这种情况下，目前的网络管理体系已经发展成一个“百家争鸣”的网络管理体系，从而导致了许多难以解决的问题。

ITU 提出了对电信网实行统一综合维护管理的 TMN 的概念，就是为了实现统一的自动化网络管理，从而适应电信网的良性发展。TMN 借鉴开放式系统互连中有关系统管理的思想及技术，通过引入管理的通用网络模型的概念，采用通用的信息模型和标准接口，以实现对各种完全不同的设备的通用管理。

TMN 的基本原理之一是使管理功能与电信功能分离。网络管理者可以从有限的几个管理节点管理电信网络中分布的电信设备。TMN 采用了 OSI 的系统管理概念和工具，如管理者/代理的概念和管理目标的使用等。因而 TMN 可以看做应用 OSI 概念来管理电信网的电信业务的网络。也就是说，TMN 是收集、处理、传送和存储有关电信网维护、操作和管理信息的一种综合手段，为电信主管部门管理电信网起着支撑作用，即协助电信主管部门管好电信网。

图 7-10 表明了 TMN 的总体构成及与它所管理的电信网之间的关系。由图 7-10 可知，TMN 与电信网相连接。TMN 中包括数据通信网、多个操作系统、工作站及电信网的一部分。

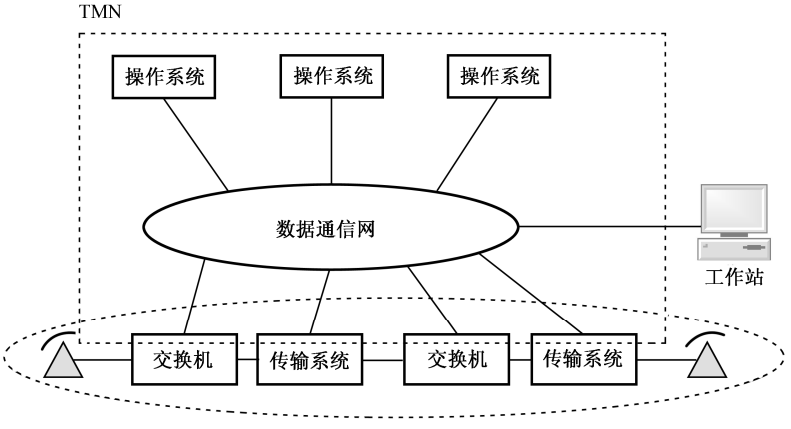


图 7-10　TMN 和电信网的关系

7.6.2　TMN 的基本概念

1．TMN 的管理功能

TMN 具有性能管理（Performance Management）、故障管理（Fault Management）、配置管理（Configuration Management）、账务管理（Accounting Management）和安全管理（Security Management）5 项功能。

（1）性能管理

性能管理主要收集通信设备网络或网络单元效能的各种统计数据，以进行规划、分析、监视、校正网络，主要任务有性能监视、性能控制和性能分析。

（2）故障管理

故障管理指能够对不正常的电信网进行环境条件检测、隔离和校正等一系列功能，诸如告警监视功能、故障定位功能、故障校正功能、测试功能。

（3）配置管理

配置管理主要实施对 NE 的控制、识别和数据交换，以及为传输网增加或去掉 NE 和通路/断路。

（4）账务管理

账务管理主要收集网络账务记录和设立使用业务的计费参数，其主要功能有计费功能和资费功能。

（5）安全管理

安全管理主要提供授权机制、访问机制、加密及密钥机制、验证机制及安全日志等功能。

2．TMN 的体系结构

依据 OSI 的管理功能分类方法的 TMN 的管理功能能适应 TMN 的需要。一般可分为五大功能域：性能管理、故障管理、配置管理、账务管理和安全管理。

（1）TMN 的功能结构

TMN 功能结构主要描述了怎样合理分布 TMN 的管理功能，怎样把一些功能单元构成具有一定管理功能的功能块，如操作系统功能（OSF）、协调功能（MF）等。同时，功能结构还描述了功能块间的接口（如 qx、q3 参考点等），利用这些功能块及参考点可以在逻辑上构成具有任意规模、任意复杂度的电信管理网。

1）TMN 功能块。

TMN 包括操作系统功能、协调功能、网元功能（NEF）、工作站功能（WSF）和 Q 适配功能（QAF）5 个功能块。

① 操作系统功能：对管理信息进行处理，执行各种管理业务及管理功能。

② 协调功能：起协调或中介作用。

③ 网元功能：主要提供通信和支持功能。

④ 工作站功能：为用户提供一种解释 TMN 信息的手段。

⑤ Q 适配功能：其任务是进行 TMN 接口与非 TMN 接口（专用接口）间的转换。

2）参考点。

参考点是表示两个功能块之间交换信息的边界点，从而区分不同的管理功能块，即表示了两个管理功能块之间进行信息交换的概念上的点。

TMN 的功能结构和参考点如**错误!未找到引用源。**所示。

TMN 有 3 类参考点，即 q 参考点、f 参考点和 x 参考点。此外，在 TMN 范畴之外，与 TMN 有关的参考点还有 g 参考点和 m 参考点。

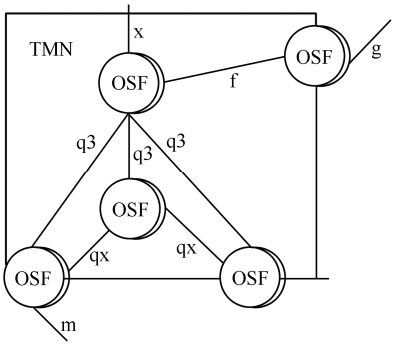


图 7-11　TMN 的功能结构和参考点

① q 参考点用来连接 OSF、QAF、MF 和 NEF。通常将连接 NEF 和 MF，QAF 和 MF，以及 MF 和 MF 的参考点称为 qx 参考点；而将连接 NEF 和 OSF，QAF 和 OSF，MF 和 OSF，以及 OSF 和 OSF 的参考点称为 q3 参考点。

② f 参考点指 OSF 和 WSF 的参考点。

③ g 参考点指用来连接用户和 WSF 的参考点，处于 TMN 之外。

④ x 参考点指连接两个 TMN 的 OSF 之间的参考点。

⑤ m 参考点指连接 QAF 和非 TMN 管理实体的参考点，也处于 TMN 之外。

参考点只是功能块之间概念上的信息交换点，只有当互连的功能块分别嵌入不同的设备时，这些参考点才称为具体的接口。

3）数据通信功能块。

数据通信功能块用于在 TMN 功能块间传递信息。它可以提供选路、转移和互通功能，涉及 OSI 的低 3 层功能。

（2）TMN 的信息结构

TMN 的信息结构主要用来描述功能模块之间交换的不同类型的管理信息的特性，其基础是管理层模型、信息模型和组织模型。所用的工具是 OSI 管理，面向对象的概念、实体关系的概念、管理者/代理的概念和电子号码簿的概念。

1）管理层次模型。

TMN 将系统管理功能分为事务管理层（BML）、业务管理层（SML）、网络管理层（NML）、网元管理层（NEML）和网元层（NEL）5 个层次，以提供在网元级、网络级、业务级和事务级上的支持。

① 网元层：管理对象的接口。

② 网元管理层：直接行使对个别网元的管理职能。

③ 网络管理层：对所辖区域内的所有网元行使管理功能。

④ 业务管理层：主要关注和负责业务。

⑤ 事务管理层：最高的逻辑功能层，负责运营商企业事务的管理。

2）信息模型。

ITU 在 M.3100 中定义的通用网络信息模型可应用于 TMN 及所有网络。

信息模型是描述管理对象及其特性的方法。它将物理资源转换成概念上的管理目标并规定目标类别、属性及其数值。事实上，它就是一种规定管理系统和管理对象之间的接口的手段。

① 网络信息模型的基本思想：网络信息模型采用面向对象的方法，对网络资源和管理方法进行抽象，从而得到一个抽象的、通用的管理标准，因可以采用抽象的、标准化的管理方法，通过对对象进行各种操作，在一个与技术无关的层次上对网络进行管理。此时，任何要管理的设备都被抽象为管理对象，并通过它们的不同属性来标识管理对象。对设备进行的管理操作就是对对象的操作，包括对对象属性值的操作，以及对对象本身的操作。对对象的管理还包括监控管理对象发出的消息，以及管理操作产生的相应行为等。

② TMN 的信息模型描述方法的几个步骤：确定完成问题所需的服务；确定完成服务所需的功能；确定支持功能的管理对象类。

③ 管理信息模型：管理信息模型分为通用管理信息模型和各个专业网使用的管理信息模型两部分。通用管理信息模型是管理对象的集合，其目的是保证不同系统之间的互操作性；不同设备使用通用的通信接口，实现与技术无关的管理；提供一个框架，使之易于扩展为技术有关的模型。

3）组织模型。

组织模型主要用来描述管理进程担任控制角色和被控制角色（即管理者和代理）的能力，以及管理者和代理之间的相互关系，同时，它也规定了 TMN 的结构及管理层内或管理层间的功能。管理者的任务是发送管理命令和接收代理发出的通知，代理的任务是管理有关管理对象、响应管理者的管理命令及向管理者发出反映管理对象行为的通知。

7.7　智能网的发展趋势

7.7.1　现有智能网技术的缺陷

随着智能网技术在各个领域的应用，它在体系结构、业务开发、系统控制、业务种类和客户化能力等重要方面体现出越来越多的不足。下面简单从体系结构、业务开发、系统控制、业务种类和客户化能力这 5 个方面分别介绍。

（1）体系结构方面

基于 ITU INCM 模型的智能网技术的发展过程以渐进的方式发展，即不同的阶段有不同的措施。针对 PSTN 的第一套智能网体系结构是 IN CS-1，网间互连及网间业务是 IN CS-2（第

二套智能网体系结构）的主要研究方向。移动网、宽带 ISDN、Internet 网与智能网的结合是 IN CS-3 和 IN CS-4 标准的研究的主要内容，还提出了基于 B-ISDN 和 Internet 的两种智能网体系结构。所以，无论在什么阶段，智能网都与具体的承载网绑定在一起，成为承载网内的实体。然而，由于不同的承载网内的智能网技术采用不同的协议，这不仅导致了不同网络的用户无法共享相同的增值业务，还为跨网络提供混合业务增加了困难。

（2）业务开发方面

在智能网中采用独立于业务的 SIB 功能块，通过将不同的 SIB 按业务逻辑连接在一起可以快速建立新的业务。但是，不同的厂商都基于不同的开发平台开发 SIB，各个厂商之间存在技术壁垒，无法使用同一的平台，这种现状限制了智能网的应用和开发。

（3）系统控制方面

传统智能网都采用了集中式控制技术，该技术适合处理业务种类和业务量相对较少的智能网。但是，随着智能网业务量的急剧增加，对智能网性能的要求越来越高，这种集中控制方式已经不能适应网络的需求，成为智能网处理能力的瓶颈，限制了智能网的发展。

（4）业务种类方面

技术支持不能开发增值业务，所以业务开发对网络运营商的依赖性越来越严重。这就导致受公众欢迎的业务由于利润少，不被网络运营商重视，所以很难推向市场。而运营商的开发人员由于不了解市场需求，开发的业务不被市场接受，最终导致了智能网的业务发展缓慢。

（5）客户化能力方面

用户的需求不同，就有不同的业务，这就是所谓的业务客户化。传统的智能网技术已经不能满足用户对业务客户化和个性化的各种需求。由于受到传统智能网基本原理的限制，虽然人们尝试解决这个问题，但仍没有显著的效果。

从当前的发展现状来看，智能网的这些缺陷已经严重阻碍了电信增值业务的进一步发展，成为困扰电信运营商的一个难题。

7.7.2 下一代智能网的重要特征

下一代智能网要能够为用户提供接入各种网络的方式和各种各样的业务，而且用户能够根据自己的需求设计自己的业务。总而言之，下一代智能网将朝着“开放性”、“分布性”和“综合性”的方向发展。

（1）开放性

对于智能网的开放性，目前没有统一的观点。下面从一些主流期刊和智能网技术相关书籍中总结出对其开放性的观点。

1）智能网的开放性表现在能够支持第三方业务开发商和提供商。

传统的智能网是专用的系统，业务逻辑往往和业务平台中的业务逻辑执行环境绑定在一起，只有业务平台的供应商或运营商才能开发基于自己业务平台的业务逻辑。而独立的业务（软件）

开发商又不愿意将自己的业务开发受限于某一个市场份额有限的智能网业务平台上。为便于业务开发，尽管业务平台的供应商提供相应的工具——SCE，而且在一定程度上加快了业务的开发进度，但没有从根本上解决如何支持第三方业务开发商的根本问题。SCE 本身也是一个专用系统，用一个智能网设备供应商提供的 SCE 开发出的业务逻辑只能在该供应商的业务平台上运行。因此，SCE 仅仅解决了设备供应商或设备运营商本身开发业务的问题。而解决这一问题可以考虑如下几个方面：统一业务执行环境，使它尽量简单、通用（如采用普通的商用操作系统或者 Java 虚拟机）；利用计算机软件领域的先进技术（如面向对象技术和面向构件技术），在通用的集成开发环境基础上，补充 SCE 需要的功能（如业务的验证和仿真），开发与增值业务相关的构件。

2）智能网的开放性表现在能够支持网络安全域以外的企业应用对网络核心资源的访问。

目前，企业应用（业务）是无法访问电信网络运营商的网络资源的。如果智能网能够支持企业应用对网络核心资源的访问，则不仅可以提高网络资源的利用率，还可以大大增强企业应用业务的功能和灵活性。解决这一问题的一种办法是在网络边沿提供能力服务器，使位于网络安全域以外的企业应用能够通过能力服务器所提供的 API 访问网络资源。

3）智能网的开放性必须保证其安全性和一致性。

网络的开放性和安全性表面上看起来是两个互相矛盾的概念，但其实它们是相互依赖，相辅相成的。没有安全性的开放式网络没有任何实用价值，但是绝对封闭安全的网络没有任何应用价值。因此，智能网应该在网络的开放性和安全性方面兼顾。

（2）分布性

传统的智能网的业务交换平台和业务控制平台本身具有分布性，一般使用一种高性能的、高可靠性的计算机系统来实现，这在智能网发展初期业务基本上可以满足用户需求。但是随着智能网业务类型的增多和业务量的增大，单个业务节点成为整个系统的瓶颈，以致于影响整个智能网的性能。

为了解决这一问题，现在的智能网基于通用分布式平台设计，该设计给予应用逻辑，各种高级的分布式功能由分布式平台来完成。在分布式计算领域，影响最大的规范有：OMG 的 CORBA，OG 的 DCE，Microsoft 的 DCOM，SUN 的 RMI。它们各有优缺点，都有各自的应用领域。在电信领域，OMG 的 CORBA 的影响最大，应用最广。智能网分布性的业务平台是基于 CORBA 平台的大型分布式处理系统。在 CORBA 平台上，不同运营商的业务平台可以进行相互协作，协同提供高级的增值业务，满足单个业务平台所无法满足的用户需求。

（3）综合性

网络技术的迅猛发展，让人无法预测未来的网络，也不知道将会有什么新标准，但是网络融合已是大趋势，网络专家普遍认为未来的网络将融合现有的各种网络，而这种融合主要体现在业务层面，未来的综合网络将能够承担各种业务，满足用户个性化、多样化的需求。

小结

智能网技术已经在全球范围内得到了快速发展，而且给人们的生活带了日新月异的变化，为人们带来了很多个性化的服务，给网络运营商和设备厂商带了巨额的利益，也丰富了人们的生活。

本章的目的使读者掌握智能网的基本概念、基本特点、网络结构、与其他网络的结合及发展趋势。

参考文献

[1] 龚双瑾，刘多．移动与 IP 智能网[M]. 北京：人民邮电出版社，2004.
[2] 杨放春，孙其博．智能网技术及其发展（修订版）[M]. 北京：北京邮电大学出版社，2002.
[3] 廖建新，龙元香，王晶，等．宽带智能网[M]. 北京：人民邮电出版社，2001.
[4] 王柏．智能网教程[M]. 北京：北京邮电大学出版社，2000.
[5] 廖建新．移动智能网[M]. 北京：北京邮电大学出版社，2000.
[6] 李晓峰．智能网技术[M]. 北京：北京邮电大学出版社，1998.

第 8 章

计算机网络编程基础

8.1 线程同步与异步套接字编程

8.1.1 事件对象

事件对象也属于内核对象，它包含以下 3 个成员。

1）使用计数。

2）用于指明该事件是一个自动重置的事件还是一个人工重置的事件的布尔值。

3）用于指明该事件处于已通知状态还是未通知状态的布尔值。

事件对象有两种不同的类型：人工重置的事件对象和自动重置的事件对象。当人工重置的事件对象得到通知时，等待该事件对象的所有线程均变为可调度线程。当一个自动重置的事件对象得到通知时，等待事件对象的线程中只有一个线程变为可调度线程。

1．创建事件对象

在程序中可以通过 CreateEvent 函数创建或打开一个命名的或者匿名的事件对象，该函数的原型声明如下所示。

```
HANDLE CreateEvent{
    LPSECURITY_ATTRIBUTES lpEventAttributes,
    BOOL bMannualReset,
    BOOL bInitialState，
    LPCTSTR lpName
};
```

该函数有 4 个参数，各参数含义如下。

1）lpEventAttributes：指向 SECURITY_ATTRIBUTES 结构体指针。如果其值为 NULL，则使用默认的安全性。

2）bManualReset：BOOL 类型，指定创建的是人工重置事件对象，还是自动重置事件对象。如果此参数为 TRUE，则表示该函数将创建一个人工重置事件对象；如果此参数为 FALSE，则

表示该参数将创建一个自动重置事件对象。如果是人工重置事件对象，当线程等到该对象的所有权之后，需要调用 ResetEvent 函数手动地将该事件对象设置为无信号状态；如果是自动重置事件对象，当线程等到该对象的所有权之后，系统会自动将该对象设置为无信号状态。

3）bInitialState：BOOL 类型，指定事件对象的初始状态。如果此参数值为真，则该事件对象初始状态是有信号状态；否则是无信号状态。

4）lpName：指定事件对象的名称。如果此参数值为 NULL，则将创建一个匿名的事件对象。

2．设置事件对象状态

SetEvent 函数将把指定的事件对象设置为有信号状态，该函数的原型声明如下所示。

```
BOOL SetEvent(HANDLE hEvent);
```

SetEvent 函数有一个 HANDLE 类型的参数，该参数指定将要设置其状态的事件对象的句柄。

3．重置事件对象状态

ResetEvent 函数将把指定的事件对象设置为无信号状态，该函数的原型声明如下所示。

```
BOOL ResetEvent(HANDLE hEvent);
```

ResetEvent 函数有一个 HANDLE 类型的参数，该参数指定将要重置其状态的事件对象的句柄。如果调用成功，则该函数返回非 0 值；否则返回 0 值。

4．利用事件对象实现线程同步

这里以火车票销售系统来讲解线程间的同步，但利用事件对象实现。事件对象属于内核事件。首先，新建一个空 Win32 Console Application 类型的工程，工程名命名为 Event，并为该工程添加一个 C++源文件，即 Event.cpp，然后在文件中添加具体的实现代码，代码如例 8-1 所示。

【例 8-1】

```
#include <windows.h>
#include <iostream>
      Using namespace std;
DWORD WINAPI Fun1Proc(
        LPVOID lpParameter     // thread data
);

DWORD WINAPI Fun2Proc(
        LPVOID lpParameter     // thread data
);

int tickets=100;
HANDLE g_hEvent;

void main()
{
        HANDLE hThread1;
        HANDLE hThread2;
```

```
        //创建人工重置事件内核对象
★   g_hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);
        //创建线程
    hThread1=CreateThread(NULL,0,Fun1Proc,NULL,0,NULL);
    hThread2=CreateThread(NULL,0,Fun2Proc,NULL,0,NULL);
    CloseHandle(hThread1);
    CloseHandle(hThread2);
        //使主线程睡眠 4s
    Sleep(4000);
    CloseHandle(g_hEvent);
}

  //线程 1 的入口参数
DWORD WINAPI Fun1Proc(
     LPVOID lpParameter      // thread data
)
{
    while(TRUE)
    {
            //请求事件对象
        WaitForSingleObject(g_hEvent,INFINITE);
        if(tickets>0)
        {
            Sleep(1);
            cout<<"thread1 sell ticket : "<<tickets--<<endl;
        }
        else
            break;
    }

    return 0;
}
  //线程 2 的入口参数
DWORD WINAPI Fun2Proc(
     LPVOID lpParameter      // thread data
)
{
     while(TRUE)
    {
            //请求事件对象
        WaitForSingleObject(g_hEvent,INFINITE);
        if(tickets>0)
        {
            Sleep(1);
            cout<<"thread2 sell ticket : "<<tickets--<<endl;
        }
```

```
                else
                    break;
            }
            return 0;
        }
```

例 8-1 所示代码主要由以下几部分组成。

（1）包含必要的头文件

因为程序中需要访问 Windows API 函数，因此需要包含 window.h 文件。另外，还用到了C++的标准输出函数，所以需要包含 C++的标准输入输出头文件，即 iostream。

（2）线程函数

在每个线程中都调用 WaitForSingleObject 函数请求事件对象，一旦得到事件对象，就可以进入所保护的代码中，完成销售火车票的工作。

（3）全局变量的定义

定义了两个全局变量，其中一个是整型变量 tickets，表示当前销售的票号，初始值为 100；另一个是 HANDLE 类型的变量 g_hEvent，用来保存创建的事件对象句柄。

（4）main 函数

当程序启动运行后，会产生主线程，main 函数就是主线程的入口。在这个主线程中可以创建新的线程。在上述 main 函数中，首先调用 CreateEvent 函数创建了一个事件内核对象，该函数的第一个参数设置为 NULL，使该事件对象使用进程默认的安全性；第二个参数设置为 TRUE，即创建一个人工重置的事件对象；第三个参数设置为 FALSE，即该事件对象初始处于无信号状态；最后一个参数设置为 NULL，即创建一个匿名的事件对象。调用 CreateThread 函数创建了两个新的线程，使主线程睡眠 4s。在 main 函数结束前，调用 CloseHandle 函数关闭所创建的事件对象句柄。

Build 并运行 Event 程序，将会发现线程 1 和线程 2 并没有如我们所期望的那样完成销售火车票的工作。在例 8-1 所示的 main 函数中创建事件对象时，将其初始状态设置为无信号状态，这样，当线程 1 和线程 2 请求 g_hEvent 事件对象时，因为该事件对象始终处于无信号状态，所以 WaitForSingleObject 函数将导致线程暂停，这两个线程都没有得到该事件对象，也就没有运行线程函数中 if 语句块内的代码。如果想让线程 1 或线程 2 得到 g_hEvent 事件对象的所有权，则必须将该事件对象设置为有信号状态。有两种方法可以实现这一目的：一种方法是在创建事件对象时，将 CreateEvent 函数的第三个参数设置为 TRUE，这样所创建的事件对象就处于有信号状态；另一种方法是在创建事件对象之后，通过调用 SetEvent 函数把指定的事件对象设置为有信号状态。这里采用后一种方法，在例 8-1 所示代码的 main 函数中，在创建事件对象（即★符号所示代码行）之后，添加下面的语句：

```
    SetEvent(g_hEvent);
```

再次运行 Event 程序，结果如图 8-1 所示。

可以看到，这时线程 1 和线程 2 确实销售了火车票，但是发现最后打印出号码为 0 的票号，说明程序存在问题。前面的内容曾提到，当人工重置事件对象得到通知时，等待该事件对象的所有线程均变为可调度线程。本例创建的就是一个人工重置的事件对象(g_hEvent)，当这个事件

对象变为有信号状态时，所有等待该对象的线程都变为可调度线程，也就是说，线程 1 和线程 2 可以同时运行，正因为这两个线程可以同时运行，所以对其所保护的代码来说，这两个线程可以同时执行该代码，从而导致程序打印出票号 0。既然两个线程可以同时运行，这就说明程序实现的线程间的同步失败了。究其原因，主要是因为本例创建的是人工重置的事件对象。当一个线程等待到一个人工重置的事件对象之后，这个事件对象仍然处于有信号状态，所以其他线程可以得到该事件对象，从而进入所保护的代码并执行。那么如何解决这一问题呢？读者可能会想到到这样的方法：既然人工重置事件对象在被一个线程得到之后仍处于有信号状态，那么线程在得到该事件对象之后，应立即调用 ResetEvent 函数，将该事件对象设置为无信号状态，然后对所保护的代码访问结束之后调用 SetEvent 函数，将该事件对象设置为有信号状态，这时才允许其他线程获得该事件对象的所有权。这时线程的代码如例 8-2 所示。

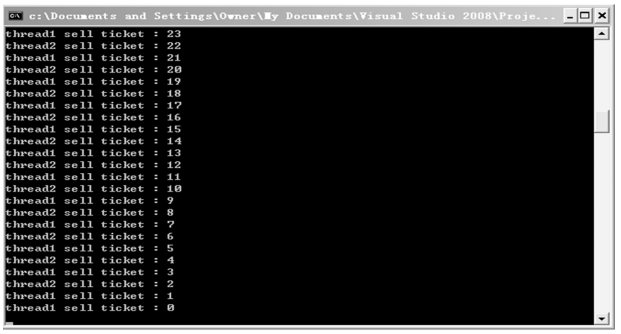


图 8-1　利用事件对象实现程同步示例（运行结果失败）

【例 8-2】

```
DWORD WINAPI Fun1Proc(
   LPVOID lpParameter
)
{
  while(TRUE)
  {
        WaitForSingleObject(g_hEvent,INFINITE);
        ResetEvent(g_hEvent);
        if(tickets>0)
        {
              Sleep(1);
              cout<<"thread1 sell ticket : "<<tickets--<<endl;
        }
        else
              break;

  }

  return 0;
}

DWORD WINAPI Fun2Proc(
   LPVOID lpParameter
)
{
```

```
    while(TRUE)
    {
        WaitForSingleObject(g_hEvent,INFINITE);
        ResetEvent(g_hEvent);
        if(tickets>0)
        {
            Sleep(1);
            cout<<"thread2 sell ticket : "<<tickets--<<endl;
        }
        else
            break;

    }

    return 0;
}
```

这样是不是就可以解决问题了呢？分析这时程序的执行过程。当线程 1 获得事件对象后，调用 ResetEvent 函数将事件对象 g_hEvent 设置为无信号状态。当进入其 if 语句之后，因为 Sleep 函数的调用，该线程睡眠，于是线程 2 开始执行。因为此时 g_hEvent 事件对象已经处于无信号状态，所以线程 2 无法请求到对象，只能一直等待。当线程 1 被唤醒之后，销售一张火车票，再调用 SetEvent 函数，将 g_hEvent 事件对象设置为有信号状态，如果这时线程 2 开始执行，则它可以得到该事件对象。同样的，在线程 2 对保护的代码执行完成之后，也调用 SetEvent 函数将事件设置为有信号状态。线程 1 又可以获得该对象，然后重复上述执行过程。

读者可以运行这时的 Event 程序，但是将会发现程序结果仍然有数值为 0 的票号，说明程序仍然没有实现线程间的同步。实际上，上述做法存在两个问题。第一个问题是，在 CPU 平台下，同一时刻只能有一个线程在运行，假设线程 1 先执行，它得到事件对象 g_hEvent，但是正好此时它的时间片终止了，于是线程 2 执行，但因为在线程 1 中，ResetEvent 函数还没有被执行，所以该事件对象仍然处于有信号状态，因此线程 2 可以得到该事件对象，也就是说，此时两个线程都可以进入所保护的代码，于是结果不可预料。另一个问题是，把 Event 程序移植到多 CPU 平台上时，线程 1 和线程 2 可以同时运行，这时再调用 SetEvent 函数将其设置为有信号状态这一操作已经不起作用，因为这两个线程都已经进入了所保护的代码，它们同时使用同一资源，结果也是未知的。所以为了实现线程间的同步，不应该使用人工重置的事件对象，而应该使用自动重置的事件对象。也就是说，应该修改例 8-1 所示代码对 CreateEvent 函数的调用，即修改例 8-1 所示代码中★符号所在的那行代码，将其第二个参数设置为 FALSE，修改结果如下：

```
    G_hEvent=CreateEvent(NULL,FALSE,FALSE,NULL);
```

这时 Event 程序的主线程将创建一个自动重置事件对象 g_hEvent，且它的初始状态为无信号状态。读者还应该将先前在线程中添加的 ResetEvent 函数和 SetEvent 函数调用（即例 8-2 所示代码中加灰显示的代码）注释起来。

再次运行 Event 程序，但是将会发现线程 1 打印出票号 100 之后，线程没有再继续运行，

程序结果如图 8-2 所示。

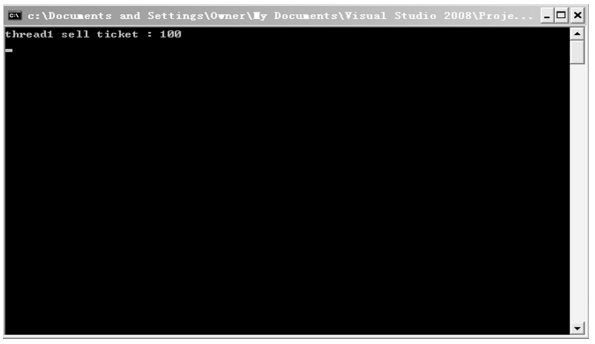


图 8-2　只有线程 1 打印票号

分析原因，前面已经提到，当一个自动重置的事件得到通知时，等待该事件的线程中只有一个线程变为可调度线程。上述结果线程 1 得到了事件对象之后，因为它是一个自动事件，所以操作系统会将该事件对象设置为无信号状态。程序继续向下执行，当线程 1 睡眠时，线程 2 执行，该线程调用 WaitForSingleObject 函数请求事件对象，然而因为这时该事件对象已经处于无信号状态，所以线程 2 只能等待，于是执行权返回给线程 1，线程 1 继续运行，输出 100。然后，线程 1 会再次请求事件对象，然而这时事件对象处于无信号状态，所以 WaitForSingleObject 函数也无法得到该事件对象，于是线程 1 只能等待。这样，线程 1 和线程 2 都在等待，所以就看到了如图 8-2 所示的结果。

因为在线程请求到事件对象后，操作系统会将该事件对象设置为无信号状态，所以为了使本程序正常运行，在线程对保护的代码访问完成之后应该立即调用 SetEvent 函数，将该事件对象设置为有信号状态，允许其他等待该对象的线程变为可调度状态。也就是说，这时线程 1 函数的代码如例 8-3 所示。

【例 8-3】

```
DWORD WINAPI Fun1Proc(
        LPVOID lpParameter
)
{
    while(TRUE)
    {
        WaitForSingleObject(g_hEvent,INFINITE);
        ResetEvent(g_hEvent);
        if(tickets>0)
        {
            Sleep(1);
            cout<<"thread1 sell ticket : "<<tickets--<<endl;
        }
        else
            break;
        SetEvent(g_hEvent);
    }
    return 0;
}
```

完成上述代码修改之后，再次运行 Event 程序，这时将会看到程序执行的结果正常了。

通过上面的例子可以知道，在使用事件对象实现线程间同步时，一定要注意区分人工重置事件对象和自动重置事件对象。当人工重置事件得到通知时，等待该事件对象的所有线程均变

为可调度线程；当一个自动重置的事件对象得到通知时，等待该事件对象的线程中只有一个线程变为可调度线程，同时，操作系统会将该事件对象设置为无信号状态，这样，当对所保护的代码执行完毕后，需要调用 SetEvent 函数将该事件对象设置为有信号状态。而人工重置的事件对象，在一个线程得到事件对象之后，操作系统并不会将该事件对象设置为无信号状态，除非显式地调用 ResetEvent 函数将其设置为无信号状态，否则该对象会一直处于有信号状态。

5．保证应用程序只有一个实例运行

通过创建一个命名的事件对象，也可以实现应用程序只有一个实例运行这一功能。对 CreateEvent 函数来说，如果创建的是命名的事件对象，并且在此函数调用之前此事件对象已经存在，那么该函数将返回已存在的这个事件对象的句柄，之后的 GetLastError 调用将返回 ERROR_ALREADY_EXISTS。因此，与利用命名互斥空间对象的方法一样，利用 CreateEvent 函数创建命名事件对象并根据其返回值判断应用程序是否已经有一个实例在运行，如果有，则应用程序退出，从而实现应用程序只有一个实例运行这一功能。

为此，需要修改 Event 程序的 main 函数，将已有的 CreateEvent 调用注释起来，然后在其后添加下述代码。

```
g_hEvent=CreateEvent(NULL,FALSE,FALSE,"tickets");
if(g_hEvent)
{
        if(ERROR_ALREADY_EXISTS==GetLastError())
        {
                cout<<"only instance can run!"<<endl;
                return;
        }
  }
```

读者可以连续运行两个 Event 程序，将会发现第二个程序运行后其窗口将打印出 “only one instance can run!”字符，并且程序随即结束。这就说明 Event 程序已经判断出当前已经有一个实例在运行，于是第二个实例退出了，从而保证程序只有一个实例在运行。

8.1.2 关键代码段及其应用

下面介绍另一种实现线程同步的方法，即利用关键代码段来实现。关键代码段也称为临界区，工作在用户方式下。它是一个小代码段，在代码段能够执行前，它必须独占对某些资源的访问权。通常把多线程中访问同一在资源的那部分代码当做关键代码段，如例 8-3 所示代码段中，线程函数 Fun1Proc 的第 9 行～第 19 行的代码就可以看做一个关键代码段。

关键代码段非常类似于我们平常使用的公用电话亭，当我们想要进入公用电话亭使用电话时，首先需要判断电话亭是否有人，如果有人正在使用电话，那么只能在电话亭外等待；当那个人使用完电话，并离开电话亭后，才能进入电话亭使用电话。同样的，当我们使用完电话后，也要离开电话亭。关键代码段的机制与此类似。就好像我们要进入电话亭使用电话这一资源时，首先需要建立一个电话亭一样，在进入关键代码段之前，首先需要初始化一个关键代码段，这

可以调用 InitializeCriticalSection 函数实现，该函数的原型声明如下。

```
Void IntializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection);
```

该函数有一个参数，是一个指向 CRITICAL_SECTION 结构体的指针。该参数是 out 类型，即作为返回值使用的。因此，在使用时，需要构造一个 CRITICAL_SECTION 结构体类型的对象，然后将该对象的地址传递给 IntializeCriticalSection 函数，系统自动维护该对象，用户不需要了解或访问该结构体对象内部的成员。

当有一个公用电话亭，并且我们想要进入该电话亭中使用电话时，首先需要判断里面有没有人。如果没有人，我们才能进去。同样，如果想要进入关键代码段，首先需要调用 EnterCriticalSection 函数，以获得指定的临界区对象的所有权，如果该所有权赋予了调用线程，则该函数返回；否则该函数会一直等待，从而导致线程等待。

当调用线程获得了指定的临界对象的所有权后，该线程就进入关键代码段，对所保护的资源进行访问。就好像公用电话亭没有人，我们就可以进去使用电话一样。当使用完电话之后，我们会离开公用电话亭。同样，线程使用完所保护的资源的之后，需要调用 LeaveCriticalSection 函数，释放指定的临界区对象的所有权，之后，其他想要获得该临界区对象所有权的线程可以获得所有权，从而进入关键代码段，访问保护的资源。

在日常生活中，当不需要某个公用电话亭时，会将其拆除。同样，对临界区对象来说，当不需要时，需要调用 DeleteCriticalSection 函数释放该对象，该函数将释放一个没有被任何线程所拥有的临界区对象的所有资源。

下面就利用关键代码段来实现线性同步。首先新建一个空间 Win32 Console Application 类型的工程，工程名取为 Critical，并为该工程添加一个 C++源文件 Critical.cpp，然后在此文件中添加具体的实现代码，可以复制前面编写的 Event.cpp 文件中的内容，并删除其中与事件对象相关的代码，然后添加使用临界区对象实现线程同步的代码，结果如例 8-4 所示。

【例 8-4】

```
#include <windows.h>
#include <iostream>
using namespace std;

DWORD WINAPI Fun1Proc(
        LPVOID lpParameter     // thread data
);

DWORD WINAPI Fun2Proc(
        LPVOID lpParameter     // thread data
);

int tickets=100;

CRITICAL_SECTION g_cs;
void main()
{
      HANDLE hThread1;
```

```
        HANDLE hThread2;
        hThread1=CreateThread(NULL,0,Fun1Proc,NULL,0,NULL);
        hThread2=CreateThread(NULL,0,Fun2Proc,NULL,0,NULL);
        CloseHandle(hThread1);
        CloseHandle(hThread2);

        InitializeCriticalSection(&g_cs);
        Sleep(4000);

        DeleteCriticalSection(&g_cs);
}

DWORD WINAPI Fun1Proc(
        LPVOID lpParameter      // thread data
)
{
        while(TRUE)
        {
                EnterCriticalSection(&g_cs);
                Sleep(1);
                if(tickets>0)
                {
                        Sleep(1);
                        cout<<"thread1 sell ticket : "<<tickets--<<endl;
                }
                else
                        break;
                LeaveCriticalSection(&g_cs);
        }

        return 0;
}

DWORD WINAPI Fun2Proc(
        LPVOID lpParameter      // thread data
)
{

        while(TRUE)
        {
                EnterCriticalSection(&g_cs);
                Sleep(1);
                if(tickets>0)
                {
                        Sleep(1);
                        cout<<"thread2 sell ticket : "<<tickets--<<endl;
```

```
                }
                else
                        break;
                LeaveCriticalSection(&g_cs);}
        cout<<"thread2 is running!"<<endl;
        return 0;
    }
```

在如例 8-4 所示代码中与 Event 程序相同的部分不再讲述，这里仅解释新添加的与关键代码段相关的代码。

因为多个线程都需要访问临界区对象，所以将它定义为全局对象，即上述代码中定义的 CRITICAL_SECTION 类型的对象 g_cs 为全局对象。

按照上面讲述的关键代码段使用的过程，在 main 函数中首先调用 InitializeCriticalSection 函数创建临界区对象，并在程序退出前，调用 DeleteCriticalSection 函数释放没有被任何线程使用的临界区对象的资源。

在主线程创建两个线程后，在进入关键代码段访问受保护的代码前，需要调用 EnterCriticalSection 函数，以判断能否得到指定临界区对象的所有权，如果无法得到该所有权，那么 EnterCriticalSection 函数会一直等待，从而导致线程暂停运行；如果能够得到该所有权，那么该线程会进入关键代码段中，访问受保护的资源。当该访问完成之后，需要调用 LeaveCriticalSection 函数，释放指定的临界区对象的所有权。

Build 并运行 Critical 程序，将会看到程序结果正常。

在使用临界区对象编程时，有一点需要注意，有时在得到临界区对象的所有权之后，可能忘记释放该所有权，这将造成什么样的后果呢？可以把例 8-4 所示代码中线程 1 函数 Fun1Proc 中的 LeaveCriticalSection 函数调用注释起来，然后再次运行 Critical 程序，将会看到始终是线程 1 在销售火车票，线程 2 没有得到销售的机会，程序结果如图 8-3 所示。

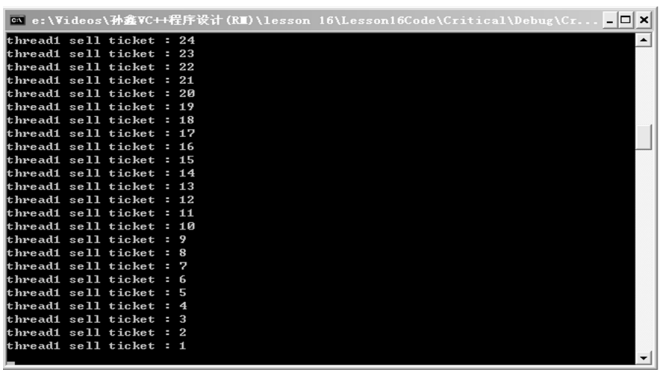


图 8-3　线程 1 没有释放临界区对象的所有权的程序结果

为了验证线程 2 确实没有得到执行关键代码段的机会，在例 8-4 所示代码的线程 2 函数 Fun2Proc 中，在 while 循环结束后，输出提示信息，即在该函数的 return 语句之前添加下面的语句：

```
    cout<<"Thread2 is running!"<<endl;
```

如果线程 2 得到了执行关键代码段的机会，它会输出“Thread2 is running!”。

再次运行 Critical 程序，将会发现始终没有看到“Thread2 is running!”这句话，这说明线程 2 始终没有得到执行关键代码段的机会。这主要是因为线程 1 获得了临界区对象的所有权，虽然线程 1 执行完成之后就退出了，但是因为该线程一直未释放临界区对象的所有权，导致线程 2 始终无法得到该所有权，只能一直等待，无法执行下面的关键代码段，进程退出时，该线程也退出了。这就好像在公用电话亭使用电话时，当一个人打完电话离开电话亭了，但由于某种原因，他把电话锁起来了，这时其他想要进入该电话亭使用电话的人始终判断该电话亭有人，无法进入。同样的，这时虽然线程 1 的运行已经终止并退出了，但线程 2 仍然无法得到运行的机会。

下面介绍关于程序死锁问题在关键代码段中的应用。

有一个哲学家进餐的问题能够很好地描述线程死锁。有多位哲学家一起用餐，但每人只有一支筷子。这时，如果有一位哲学家能将他的那只筷子交出来，让其他哲学家先吃，之后，再将一双筷子交回来，这样所有哲学家能够吃到食物了。但是哲学家们担心把筷子交给别人先吃，那么别人吃完食物之后，如果不把筷子交回来，自己不就吃不到食物了吗？所以他们都希望其他人先把筷子交出来，让自己先吃。然而由于每位哲学家都这样想，从而导致每位哲学家都不肯交出筷子，于是所有的哲学家都吃不到美食。这就是一个死锁的实例。

对多线程来说，如果线程 1 拥有了临界区对象 A，等待临界区对象 B 的所有权，线程 2 拥有了临界区对象 B，等待临界区对象 A 的所有权，这就造成了死锁。下面通过代码来演示线程死锁的发生。在已有的 Critical 程序上进行修改，结果如例 8-5 所示。

【例 8-5】

```
#include <windows.h>
#include <iostream>
using namespace std;

DWORD WINAPI Fun1Proc(
        LPVOID lpParameter      // thread data
);

DWORD WINAPI Fun2Proc(
        LPVOID lpParameter      // thread data
);

int tickets=100;

CRITICAL_SECTION g_csA;
CRITICAL_SECTION g_csB;

void main()
{
        HANDLE hThread1;
        HANDLE hThread2;
```

```
    hThread1=CreateThread(NULL,0,Fun1Proc,NULL,0,NULL);
    hThread2=CreateThread(NULL,0,Fun2Proc,NULL,0,NULL);
    CloseHandle(hThread1);
    CloseHandle(hThread2);

    InitializeCriticalSection(&g_csA);
    InitializeCriticalSection(&g_csB);
    Sleep(4000);

    DeleteCriticalSection(&g_csA);
    DeleteCriticalSection(&g_csB);
}

DWORD WINAPI Fun1Proc(
     LPVOID lpParameter      // thread data
)
{
    while(TRUE)
    {
        EnterCriticalSection(&g_csA);
        Sleep(1);
        EnterCriticalSection(&g_csB);
        if(tickets>0)
        {
            Sleep(1);
            cout<<"thread1 sell ticket : "<<tickets--<<endl;
        }
        else
            break;
        LeaveCriticalSection(&g_csB);
        LeaveCriticalSection(&g_csA);
    }

    return 0;
}

DWORD WINAPI Fun2Proc(
     LPVOID lpParameter      // thread data
)
{

    while(TRUE)
    {
        EnterCriticalSection(&g_csB);
        Sleep(1);
        EnterCriticalSection(&g_csA);
```

```
            if(tickets>0)
            {
                Sleep(1);
                cout<<"thread2 sell ticket : "<<tickets--<<endl;
            }
            else
                break;
            LeaveCriticalSection(&g_csA);
            LeaveCriticalSection(&g_csB);
        }
        cout<<"thread2 is running!"<<endl;
        return 0;
    }
```

首先，例 8-5 所示代码创建了两个临界区对象：g_csA 和 g_csB。在程序中，如果需要对某一种资源进行保护，则可以构建一个临界区对象。这与现实生活中的情况是一样的，如在程序中想要保护电话这种资源，则构建一个临界区对象来实现，就好像建立一个电话亭一样；如果在程序中还想访问自动柜员机这种资源，则可以再创建一个临界区对象，对自动柜台机资源进行保护。

例 8-5 所示代码在 main 函数中，调用 InitializeCriticalSection 函数对新创建的两个临界区对象 g_csA 和 g_csB 都进行了初始化，并在程序退出前调用 DeleteCriticalSection 函数释放这两个临界区对象的所有资源。

然后，在线程 1 中调用 EnterCritical 函数请求临界区对象 g_csA 的所有权，当得到所有权后，再请求临界区对象 g_csB 的所有权。当线程 1 访问完保护的资源后，调用 LeaveCriticalSection 函数释放两个临界区对象的所有权。注意因为调用 LeaveCriticalSection 函数时，该函数会立即返回，并不会导致线程等待，所以临界区对象所有权的顺序是无所谓的。

对线程 2 来说，它先请求临界区对象 g_csB 的所有权，然后等待临界区对象 g_csA 的所有权。在访问保护的资源之后，释放所有临界区对象的所有权。

下面，我们来分析上述程序的执行过程。当线程 1 得到临界区对象 g_csA 的所有权之后，调用 Sleep 函数，该线程将睡眠 lms，放弃执行机会。于是，操作系统会选择线程 2 执行，该线程首先等待的是临界区对象 g_csB 的所有权，当它得到该所有权之后，调用 Sleep 函数，使线程 2 也睡眠 1ms。于是线程 1 执行，这时它需要等待临界区对象 g_caB 的所有权。然而这时临界区对象 g_csB 已经被线程 2 所拥有，因此线程 1 会等待。当线程 1 等待时，线程 2 执行，这时它需要等待临界区对象 g_csA 的所有权。然而临界区对象 g_csA 的所有权已经被线程 1 所拥有，因此线程 2 也进入等待状态。这样就导致线程 1 和线程 2 都在等待对方交出临界区对象的所有权，于是就造成了死锁。

我们可以运行这时的 Critical 程序，将会看到如图 8-4 所示的结果。可以看到，线程 1 和线程 2 都没有执行关键代码段中的代码，说明它们都没有得到所需的临界区对象的所有权。

图 8-4　线程死锁结果

因此，在利用多线程技术编写程序的过程中，在实现线程同步时一定要多注意，应该避免发生线程死锁。

8.1.3　基于消息的异步套接字

Windows 套接字在两种模式下执行 I/O 操作：阻塞模式和非阻塞模式。在阻塞模式下，在 I/O 操作完成前，执行操作的 Winsock 函数会一直等待，不会立即返回（即不会将控制权交还给程序），如程序中调用了 recvfrom 函数后，如果这时网络上没有数据传送过来，该函数就会阻塞程序的执行，从而导致调用线程暂停运行。网络聊天程序就是在阻塞模式下，为了接收数据而单独创建了一个线程，在该线程中调用 recvfrom 函数接收数据，如果网络上没有数据传过来，该函数就会阻塞，从而导致所创建的那个线程暂停运行，但是并不会影响主线程的运行。而在非阻塞模式下，Winsock 函数无论如何都会立即返回，在该函数执行的操作完成之后，系统会采用某种方式将操作结果通知给调用线程，后者根据通知信息可以判断该操作是正常完成了，还是出现了错误。

因为在很多情况下，阻塞方式会影响应用程序的性能，所以有时需要采用非阻塞模式实现网络应用程序，有多种机制可以实现这种方式。Windows Sockets 为了支持 Windows 消息驱动机制，使应用程序开发者能够方便地处理网络通信，对网络事件采用了基于消息的异步存取策略。Windows Sockets 的异步选择函数 WSAAsyncSelect 提供了消息机制的网络事件选择，当使用它登记的网络事件发生时，Windows 应用程序相应的窗口函数将收到一个消息，消息中指示了发生的网络事件，以及与该事件相关的一些信息。

因此，可以针对不同的网络事件进行登记，如果登记一个网络读取事件，一旦有数据到来，就会触发这个事件，操作系统就会通过一个消息来通知调用线程，后者可以在相应的消息响应函数中接收此数据。因为是在该数据到来之后操作系统发出的通知，所以这时肯定能够接收到数据。采用异步套接字能够有效地提高应用程序的性能。

1．相关函数说明

（1）WSAAsyncSelect 函数

函数原型声明如下。

```
Int WSAAsyncSelect(SOCKET s, HWND hWnd, unsigned int wMsg, long lEvent);
```

该函数为指定的套接字请求基于 Windows 消息的网络事件通知，并自动将该套接字设置为非阻塞模式。该函数有 3 个参数，其含义分别如下所述。

1）s：标识请求网络事件通知的套接字。

2）hWnd：标识一个网络事件发生时接收消息的窗口的句柄。

3）wMsg：指定网络事件发生时窗口将接收的消息。

4）lEvent：用于指定应用程序感兴趣的网络事件，该参数可以是表 8-1 中列出的值之一，并且可以采用位或操作来构造多个事件。

表 8-1　lEvent 参数的取值和说明

取　　值	说　　明
FD_READ	应用程序想要接收相关是否可读的通知，以便读取数据
FD_WRITE	应用程序想要接收相关是否可写的通知，以便发送数据
FD_OOB	应用程序想要接收是否带外(OOB)数据抵达的通知
FD_ACCEPT	应用程序想要接收与进入连接相关的通知
FD_CONNECT	应用程序想要接收与连接有关的通知
FD_CLOSE	应用程序想要接收与套接字关闭有关的通知
FD_QOS	应用程序想要接收套接字“服务质量”发生更改的通知
FD_GROUP_QOS	应用程序想要接收套接字组“服务质量”发生更改的通知
FD_ROUTING_INTERFACE_CHANGE	应用程序想要接收在指定的方向上，与路由接口发生变化有关的通知
FD_ADDRESS_LIST_CHANGE	应用程序想要接收针对套接字的协议簇，本地地址列表发生变化的通知

（2）WSAEnumProtocols 函数

函数原型声明如下。

```
Int WSAEnumProtocols(LPINT lpiProtocols, LPWSAPROTOCOL_INFO lpProtocolBuffer, ILPDWORD lpdwBufferLength);
```

Win32 平台支持多种不同的协议，采用 Winsock2 可以编写可直接使用任何一种协议的网络应用程序。通过 WSAEnumProtocols 函数可以获得系统中安装的网络协议的相关信息。该函数各参数的含义如下。

1）lpiProtocols：一个以 NULL 结尾的协议标识号数组。这个参数是可选的，如果 lpiProtocols 为 NULL，则 WSAEnumProtocols 函数将返回所有可用协议的信息，否则只返回数组中列出的协议信息。

2）lpProtocolBuffer：out 类型的参数，作为返回值使用，一个用 WSAPROTOCOL_INFO 结构体填充的缓存区。WSAPROTOCOL_INFO 结构体用来存放或得到一个指定协议的完整信息。

3）lpdwBufferLength：in/out 类型的参数。在输入时，指定传递给 WSAEnumProtocols 函数 lpProtocolBuffer 缓冲区的长度；在输出时，存有获取请求信息需传递给 WSAEnumProtocols 函数的最小缓冲区长度。

WSAEnumProtocols 函数不能重复调用，传入的缓冲区必须足够大以便存放所有的元素。

这个规定降低了该函数的复杂度，并且由于一个机器上装载的协议数目很少，因此并不会产生问题。

（3）WSAStartup

WSAStartup 函数将初始化进程使用的 WS2_32.DLL，该函数原型声明如下所示。

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

WSAStartup 函数有两个参数，其含义分别如下。

1）wVersionRequested：调用进程可以使用的 Windows Sockets 支持的最高版本。该参数的高位字节指定 Winsock 库的副版本，而低位字节是主版本。

2）lpWSAData：out 类型的参数，作为返回值使用，是一个指向 WSADATA 数据结构类型变量的指针，用来接收 Windows Sockets 实现的细节。

（4）WSACleanup

WSACleanup 函数将终止程序对套接字库(WS2_32.DLL)的使用。该函数的原型声明如下。

```
int WSACleanup(void);
```

（5）WSASocket

Winsock 库中的扩展函数 WSASocket 将创建套接字，其原型声明如下。

```
SOCKET WSASocket(int af, int type, int protocol, LPWSAPROTOCOL_INFO lpProtocolInfo, GROUP g, DWORD dwFlags);
```

该函数前 3 个参数和前面介绍过的 socket 函数的前 3 个参数含义相同，其他参数的含义如下。

1）lpProtocolInfo：一个指向 LPWSAPROTOCOL_INFO 结构体的指针，该结构定义了所创建的套接字的特性。如果 lpProtocolInfo 为 NULL，则 WinSock2.DLL 使用前 3 个参数来决定使用哪一个服务提供者，它选择能够支持规定的地址簇、套接字类型和协议值的第一个传输提供者。如果 lpProtocolInfo 不为 NULL，则套接字绑定到指定的结构 WSAPROTOCOL_ INFO 相关的提供者。

2）g：保留。

3)dwFlags：指定套接字属性的描述。如果该参数的取值为 WSA_FLAG_ OVERLAPPED，则创建一个重叠套接字，这种类型的套接字后续的重叠操作与前面讲述的文件的重叠操作是类似的。随后在套接字上调用 WSASend，WSARecv，WSASendTo，WSARecvFrom，WSAIoctl，这些函数都会立即返回。在这些操作完成之后，操作系统会通过某种方式来通知调用线程，后者可以根据信息判断操作是否完成。

（6）WSARecvFrom 函数

WSARecvFrom 函数原型如下。

```
WSARecvFrom(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumberOfBytesRecvd, LPDWORD lpFlags, struct sockaddr FAR *lpFrom, LPINT lpFromlen, LPWSAOVERLAPPED lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);
```

WSARecvFrom 函数接收数据报类型的数据，并保存数据发送方的地址。该函数有 9 个参数，其含义分别如下。

1）s：标识套接字描述符。

2）lpBuffers：in/out 类型的参数，一个指向 WSABUF 结构体数组的指针，该结构体的定义如下。

```
typedef struct __WSABUF{
    u_long len;       //buffer length
    char FAR *buf;       //pointer to buffer
}WSABUF，FAR *LPWSABUF;
```

每一个 WSABUF 结构体都包含一个指定缓冲区的指针（buf 成员）和该缓冲区的长度（len）。

3）dwBufferCount：lpBuffers 数组中 WSABUF 结构体的数目。

4）lpNumberOfBytesRecvd：out 类型的参数，如果接收操作立即完成，则该参数是一个指向本次调用所接收的字节数的指针。

5）lpFlags：in/out 类型的参数，一个指向标志位的指针，这些标志将会影响函数的行为。该参数的取值如表 8-2 所示，可以利用位或操作将这些标志组合起来使用。

表 8-2　lpFlags 参数取值

标　志	说　明
MSG_PEEK	浏览到来的数据。这些数据被复制到缓冲区中，但并不从输入队列中移除。此标志仅对非重叠套接字有效
MSG_OOB	处理带外数据
MSG_PARTIAL	此标志仅用于面向消息的套接字。作为输出参数时，此标志表明数据是发送方传送消息的一部分。消息的剩余部分将在随后的接收操作中被传送。如果随后的某个接收操作没有此标准，则表明这是发送方发送的消息的尾部。作为输入参数时，此标志表明接收操作应该是完成的，即使只是一条消息的部分数据已被服务提供者所接收

6）lpFrom：out 类型的参数，是一个可选的指针，指向重叠操作完成后存放源地址的缓冲区。

7）lpFromlen：in/out 类型的参数，一个指向 lpFrom 指向的缓冲区大小的指针，仅当指定了 lpFrom 参数时才需要使用这个参数。

8）lpOverlapped：一个指向 LPWSAOVERLAPPED 结构体的指针（对于非重叠套接字则忽略）。

9）lpCompletionRoutine：一个指向接收操作完成时调用的完成例程的指针（对于非重叠套接字则忽略）。

如果创建的是重叠套接字，在使用 WSARecvFrom 函数时，一定要注意最后两个参数的值。因为这时将采用重叠 IO 操作，WSARecvFrom 函数会立即返回。当接收数据这一操作完成之后，操作系统会调用 lpCompletionRoutine 参数指定的例程通知调用线程，这个例程实际上就是一个回调函数，该函数的原型声明如下。

```
void CALLBACK CompletionROUTINE(IN DWORD dwError, IN DWORD cb Transferred, IN LPWSAOVERLAPPED lpOverlapped, IN DWORD dwFlags);
```

通过 WSARecvFrom 函数的第 2 个参数，可以知道，在调用该函数接收数据时，可以同时指定多个 WASBUF 结构体变量，一起接收数据，并通过该函数的第 3 个参数指定 WSABUF 结构体的数量。

为什么要定义多个 WSABUF 结构体变量同时接收数据呢？是数据量太大吗？当然不是，如果数据量太大，定义一个大容量的数据缓冲区进行一次接收操作即可，并没有必要定义多个缓冲区。那么同时指定多个缓冲区有什么好处呢？假如我们要开发一个防火墙产品，其中包括防火墙管理中心，既然作为管理中心，就需要接收防火墙日志信息，而日志信息的格式通常是自定义的，也就是在一连串的字节流中，人为地定义前几个字节、中间几个字节，以及最后几个字节各自表示的含义，如果采用 WSARecvFrom 函数接收数据，则可以针对传送的信息，分别提供不同的缓冲区去接收，然后提供多个缓冲区同时接收数据。

（7）WSASendTo 函数

WSASendTo 函数原型声明如下。

```
int WSASendTo(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumberofBytesSent, DWORD dwFlags, const struct sockaddr FAR *lpTo, int iToLen, LPWSAOVERLAPPED lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionROUTINE);
```

1）s：标识一个套接字（可能已连接）的描述符。

2）lpBuffers：一个指向 WSABUF 结构体的指针。每一个 WSABUF 结构体包含一个缓冲区的指针和缓冲区的长度。

3）dwBufferCount：lpBuffers 数组中 WSABUF 结构体的数目。

4）lpNumberofBytesSent：out 类型的参数，如果发送操作立即完成，则为一个指向本次调用所发送的字节数的指针。

5）dwFlags：指示影响操作行为的标识位。本例设置为 0 即可。

6）lpTo：可选指针，指向目标套接字的地址。

7）iToLen：lpTo 中地址的长度。

8）lpOverlapped：一个指向 WSAOVERLAPPED 结构的指针（对于非重叠套接字则忽略）。

9）lpCompletionROUTINE：一个指向接收操作完成时调度的完成例程的指针（对于非重叠套接字则忽略）。

2．套接字库在网络聊天室程序的实现

下面采用基于消息的异步套接字实现网络聊天程序。首先，新建一个基于对话框的工程，工程名为 Chat，并将该对话框资源上自动创建的控制全部删除，然后添加一些控制。

下面介绍加载套接字库及其简单应用。

与前面使用套接字编程一样，首先需要加载套接字库并进行版本协商。前面我们使用了 MFC 函数 AfxSocketInit 来完成这一任务，但是该函数只能加载 1.1 版本的套接字库，本程序中需要使用套接字 2.0 中的一些函数，因此应该调用 WSAStartup 函数初始化程序所使用的套接字库。同样，在应用程序类时，即在 CChatApp 类的 InitInstance 函数中加载套接字，因此，在该函数的开始位置添加如例 8-6 所示的代码。

【例 8-6】

```
WORD wVersionRequested;
WSADATA wsaData;
int err;
```

```
    wVersionRequested = MAKEWORD( 2, 2 );

    err = WSAStartup( wVersionRequested, &wsaData );
    if ( err != 0 ) {

        return FALSE;
    }
    if ( LOBYTE( wsaData.wVersion ) != 2 ||
         HIBYTE( wsaData.wVersion ) != 2 ) {

        WSACleanup( );
        return FALSE;
    }
```

如果调用成功，WSAStartup 函数将返回 0；否则，将返回一个预定义的错误代码。因此，在例 8-6 所示的代码中，对此函数的返回值进行检测，如果不是 0，则说明 WSAStartup 函数调用失败，InitInstance 函数立即返回 FALSE 值。

本例加载 2.2 版本的 Winsock 库，如果版本不符合要求，调用 WSACleanup 函数终止对套接字库的使用，然后 InitInstance 函数立即返回 FALSE 值。

同样，因为调用了 Winsock 2.0 中的函数，所以需要包含相应的头文件：winsock2.h。与前面的 Chat 程序一样，将该文件的包含语句放在 stdafx.h 文件中，即在文件中添加下面的语句：

```
#include <winsock2.h>
```

当然，还需要为 Chat 工程链接 ws2_32.lib 文件。

创建并初始化套接字，实现步骤与前面的 Chat 程序一样，为 CChatDlg 类增加一个 SOCKET 类型的成员变量 m_socket，即套接字描述符，并将其访问权限设置为私有。然后为 CChatDlg 类添加一个 BOOL 类型的成员函数 InitSocket，用来初始化该类的套接字成员变量，该函数的实现代码如例 8-7 所示。

【例 8-7】

```
BOOL CChatDlg::InitSocket()
{
    m_socket=WSASocket(AF_INET,SOCK_DGRAM,0,NULL,0,0);
    if(INVALID_SOCKET==m_socket)
    {
        MessageBox("创建套接字失败！");
        return FALSE;
    }
    SOCKADDR_IN addrSock;
    addrSock.sin_addr.S_un.S_addr=htonl(INADDR_ANY);
    addrSock.sin_family=AF_INET;
    addrSock.sin_port=htons(6000);
    if(SOCKET_ERROR==bind(m_socket,(SOCKADDR*)&addrSock,sizeof(SOCKADDR)))
    {
        MessageBox("绑定失败！");
        return FALSE;
```

```
        }
        if(SOCKET_ERROR==WSAAsyncSelect(m_socket,m_hWnd,UM_SOCK,FD_READ))
        {
            MessageBox("注册网络读取事件失败！");
            return FALSE;
        }

        return TRUE;
    }
```

前面示例中使用的是 socket 函数创建套接字，这里使用 Winsock 库中的扩展函数 WSASocket 来完成这一功能。

在例 8-7 所示代码中，对 WSASocket 函数的返回值进行判断，如果是 INVALID_SOCKET，则说明该函数调用失败，提示用户“创建套接字失败！”，并使 InitSocket 函数立即返回，返回值是 FALSE。

如果套接字创建成功，则要将该套接字绑定到某个 IP 地址和端口上，Winsock 2.0 的库中没有提供 bind 函数的扩展函数，所以这里仍然使用该函数来完成套接字的绑定。首先定义一个地址结构体（SOCKADDR_IN）变量 addSock，并为其成员赋值。然后调用 bind 函数，并对其返回值进行判断，如果是 SOCKET_ERROR，则说明 bind 函数调用出错了，提示用户“绑定失败”。

可以调用 WSAAsyncSelect 函数请求一个基于 Windows 消息的网络事件通知，该函数的第一个参数就是标识请求网络事件通知的套接字描述符；本例使对话框窗口接收消息，因此第 2 个参数就是该对话框的窗口句柄，即 CChatDlg 类的 m_hWnd 成员；第 3 个参数指定了一个自定义的消息（UM_SOCK），一旦指定的网络事件发生，操作系统就会发送该自定义的消息通知调用线程；第 4 个参数是注册的事件，本例注册了一个读取事件（FD_READ）。这样，一旦有数据到来，就会触发 FD_READ 事件，系统会通过 UM_SOCK 消息来通知调用线程，于是，即可在该消息的响应函数中接收数据。上述代码中，对 WSAAsyncSelect 函数的返回值做判断，如果函数调用失败，则提示用户“注册网络读取事件失败！”，并返回 FALSE 值。

如果上述操作全部成功，则 InitSocket 函数返回 TRUE。这就是在 InitSocket 函数对套接字进行初始化的处理过程。可以在 CChatDlg 类的 OnInitDialog 函数中调用这个函数，以便使程序完成套接字的初始化工作。因此，在 OnInitDialog 函数最后的 return 语句之前添加下面的语句：

```
    InitSocket();
```

然后，在 CChatDlg 类的头文件中，定义自定义消息 UM_SOCK，定义代码如下。

```
    #define UM_SOCK     WM_USER+1
```

下面介绍接收端功能的实现。这里应该注意，在注册的事件发生后，操作系统向调用进程发送相应的消息时，还会将该事件相应的信息一起传递给调用进程，这些信息是通过消息的两个参数传递的。因此在定义 UM_SOCK 消息响应函数时应带有 wParam 和 lParam 两个参数。在 CChatDlg 类的头文件中添加例 8-8 所示代码中 UM_SOCK 消息响应函数原型的声明。

【例 8-8】

```
    //Generated message map functions
```

```
    //{{AFX_MSG(CChatDlg)
    Virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    //}}AFX_MSG
    afx_msg void_OnSock(WPARAM,LPARAM);
    DECLEAR_MESSAGE_MAP()
```

在 CChatDlg 类的源文件中添加 UM_SOCK 消息映射，如例 8-9 所示。

【例 8-9】

```
BEGIN_MESSAGE_MAP(CChatDlg, CDialog)
    //{{AFX_MSG_MAP(CChatDlg)
     ON_WM_SYSCOMMAND()
     ON_WM_PAINT()
     ON_WM_QUERYDRAGICON()
     ON_BN_CLICKED(IDC_BTN_SEND, OnBtnSend)
     //}}AFX_MSG_MAP
     ON_MESSAGE(UM_SOCK,OnSock)
END_MESSAGE_MAP()
```

最后是该消息响应函数的实现，具体代码如例 8-10 所示。

【例 8-10】

```
void CChatDlg::OnSock(WPARAM wParam,LPARAM lParam)
{
     switch(LOWORD(lParam))
     {
      case FD_READ:
          WSABUF wsabuf;
          wsabuf.buf=new char[200];
          wsabuf.len=200;
          DWORD dwRead;
          DWORD dwFlag=0;
          SOCKADDR_IN addrFrom;
          int len=sizeof(SOCKADDR);
          CString str;
          CString strTemp;
          HOSTENT *pHost;
          if(SOCKET_ERROR==WSARecvFrom(m_socket,&wsabuf,1,&dwRead,&dwFlag,
                              (SOCKADDR*)&addrFrom,&len,NULL,NULL))
          {
              MessageBox("接收数据失败！");
              return FALSE;
          }
          pHost=gethostbyaddr((char*)&addrFrom.sin_addr.S_un.S_addr,4,AF_INET);
          str.Format("%s 说:%s",inet_ntoa(addrFrom.sin_addr),wsabuf.buf);
          str.Format("%s 说:%s",pHost->h_name,wsabuf.buf);
```

```
                str+="\r\n";
                GetDlgItemText(IDC_EDIT_RECV,strTemp);
                str+=strTemp;
                SetDlgItemText(IDC_EDIT_RECV,str);
                break;
            }
        }
```

读者可能会认为，既然我们请求了一个基于 Windows 消息的网络事件通知，即只请求了 FD_READ 这种事件，那么在 OnSock 函数中就可以直接调用 WSARecvFrom 函数接收数据了。这种想法本身并没有错误。但是需要注意，在基于套接字请求网络事件通知时，可以同时请求多个网络事件，也就是说，不但可以请求 FD_READ 网络读取事件，还可以请求 FD_WRITE 网络写入事件，那么在对指定的消息进行处理时，即在相应的消息响应函数中，可以根据当前发生的网络事件进行相应的处理。在本例中，因为只请求了 FD_READ 网络读取事件，所以 OnSock 函数可以直接调用 WSARecvFrom 函数接收数据。但这并不是一种良好的代码设计风格。作为一种良好的代码设计风格，应该在此函数中判断是否是网络读取事件发生了，如果是，则读取数据。

一旦网络事件发生，系统会发送自定义的消息通知调用线程，随该消息发送的信息是随着消息的两个参数发生的。当一个指定的网络事件在指定的套接字上发生时，应用程序窗口接收到指定的消息，该消息的 wParam 参数标识已经发生网络事件的套接字，即 lParam 参数的低位字，即可知道当前发生的网络类型。

因此在例 8-10 所示代码的 OnSock 函数中，首先使用 switch 语句，利用 LOWORD 宏取出 lParam 参数的低位字，并判断是否是网络读取事件发生了，如果是，则调用 Winsock 库的扩展函数 WSARecvFrom 接收数据。在调用该函数之前，先根据其需要的参数定义一些变量。首先定义了一个 WSABUF 结构体类型的变量 wsabuf，并为其赋值，将其缓冲区设置为 200 字节；然后，定义一个 dwRead 变量，用来保存实际接收到的数据长度；再定义一个 SOCKADDR_IN 结构体变量 addrFrom，用来接收发送方的地址信息，接收定义的一个变量 len，并用 SOCKADDR_IN 结构体长度对其初始化。最后，调用 WSWRecvFrom 函数接收数据，第 1 个参数是套接字描述符；第 2 个参数指定接收数据的 WSABUF 结构体数组；第 3 个参数指定用来接收数据的 WSABUF 结构体变量的数目；第 4 个参数保存实际接收到的数据；第 5 个参数是标识位，本例指定为 0 即可；第 6 个参数是用来接收发送方地址信息的地址结构体指针；第 7 个参数是上一参数的长度；本例将最后两个参数都设置为 NULL。另外，因为数据接收操作是在网络读取事件发生时进行的，所以一般这种读取操作会成功，但是为了代码风格统一，在程序中也可以对 WSARecvFrom 函数的返回值进行判断，如果出错，则利用 MessageBox 函数显示一个消息框，提示用户“接收数据失败！”，然后释放已分配的内容，直接返回。

例 8-10 所示代码中的 OnSock 函数在接收数据后，将该数据进行格式化。首先，取出发送端地址（保存在 addrFrom 变量的 sin_addr 成员中），并将它转换为点分十进制表示的字符串；其次，从对话框的接收端编辑框中取出已有的数据，保存到 strTemp 变量中；再次，将当前接收到的数据加上回车换行符后，再加上 strTemp 变量中保持的以前接收到的数据，调用 SetDlgItemText 函数将所有数据都放置到接收编辑框上，最后，一定不要忘记释放已分配的内存。

另外，为了让接收编辑框控制支持回车换行，还需要设置其 Multiline 属性。以上就是数据的接收部分。

编写发送端程序。双击 Chat 程序主界面对话框资源上的“发送”按钮，VC++开发环境将为该按钮自动生成一个按钮单击命令响应函数 OnBtnSend，然后在此函数中添加代码实现数据发送的功能，结果如例 8-11 所示。

【例 8-11】

```
void CChatDlg::OnBtnSend()
{
    // TODO: Add your control notification handler code here
    DWORD dwIP;
    CString strSend;
    WSABUF wsabuf;
    DWORD dwSend;
    int len;
    CString strHostName;
    SOCKADDR_IN addrTo;
    HOSTENT* pHost;
    if(GetDlgItemText(IDC_EDIT_HOSTNAME,strHostName),strHostName=="")
    {
        ((CIPAddressCtrl*)GetDlgItem(IDC_IPADDRESS1))->GetAddress(dwIP);
        addrTo.sin_addr.S_un.S_addr=htonl(dwIP);
    }
    else
    {
        pHost=gethostbyname(strHostName);
        addrTo.sin_addr.S_un.S_addr=*((DWORD*)pHost->h_addr_list[0]);
    }
    addrTo.sin_family=AF_INET;
    addrTo.sin_port=htons(6000);
    GetDlgItemText(IDC_EDIT_SEND,strSend);
    len=strSend.GetLength();
    wsabuf.buf=strSend.GetBuffer(len);
    wsabuf.len=len+1;
    SetDlgItemText(IDC_EDIT_SEND,"");
    if(SOCKET_ERROR==WSASendTo(m_socket,&wsabuf,1,&dwSend,0,
            (SOCKADDR*)&addrTo,sizeof(SOCKADDR),NULL,NULL))
    {
        MessageBox("发送数据失败！");
        return;
    }
}
```

在发送数据时，首先获取 IP 地址控制的用户输入的对方 IP 地址，这可以通过调用 GetDlgItem 函数得到 IP 地址控件，然后调用该控制对象的 GetAddress 函数得到 IP 地址。

接着，定义一个地址结构（SOCKADDR_IN）变量 addTo，并设置其成员，成员 sin_family

设置为 AF_INET；成员 sin_port 是发送端端口号，因为接收端是在 6000 这个端口号等待接收数据的，所以发送端可能也需要将端口设置为 6000；成员 sin_add.S_un.S_addr 是对方的 IP 地址，并且是 DWORD 类型，虽然上面刚刚获得的 IP 地址 dwIP 也是 DWORD 类型，但是这里仍然需要调用 htonl 函数进行转换，因为这里需要的是网络字节顺序的地址，而 dwIP 变量保存的是主机字节顺序的地址。

本例发送数据类型的变量，也利用 Winsock 提供的扩展函数 WSASendTo 来实现。首先定义一个 CString 类型的变量 strSend，然后调用 GetDlgItemText 函数从编辑框上获取要发送的数据，并保存到 strSend 变量中。再定义一个变量 len，用来保存 strSend 中的数据赋给 wsabuf 变量的 buf 成员，又因为该成员的类型是 char*，而 strSend 是 CString 类型，所以可以通过 CString 类提供的 GetBuffer 函数将一个 CString 类型的对象转换为 char*类型的对象并返回。同时，给 wsabuf 的 len 成员赋值时，在发送数据的数目上多加一个字节，这主要是多传送一个'\0'字符，这样在接收端将接收到以"\0"为结尾的数据。调用 SetDlgItemText 函数将发送编辑框中的内容清空。然后，调用 WSASendTo 函数发送数据。同样，可以对该函数的返回值进行判断，如果该函数返回的是 SOCKET_ERROR，则说明发送数据操作失败了，提示用户 "发送数据失败!"，然后直接返回。

当然，与前面的 Chat 程序一样，还可以将"发送"按钮设置为默认按钮，这样用户在输入了将要发送的数据之后，只要按 Enter 键即可发送该数据。

读者可以自行测试本程序，将会发现本程序实现了前面 Chat 程序的功能。

读者应该注意到，本程序是在同一个线程中实现了接收端和发送端，如果采用阻塞套接字，可能会因为 WSARecvFrom 函数的阻塞调用而导致线程暂停运行，所以本程序采用了异步选择机制，在同一个线程中完成了接收端和发送端的功能，程序运行的效果与前面讲到采用多线程技术实现的聊天室程序结果是类似的。在编写网络应用程序时，采用异步选择机制可以提高网络应用程序的性能，如果配合多线程技术，则会大大提高所编写的网络应用程序的性能。

最后为 CChatApp 类增加一个析构函数，主要是在此函数中调用 WSACleanup 函数，终止对套接字库的使用，具体代码如例 8-12 所示。

【例 8-12】

```
CChatApp::~CChatApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
        WSACleanup();
}
```

同样，为 CChatDlg 类也提供一个析构函数，在此函数中判断 m_socket 变量是否有值，如果该套接字有值，则调用 closesocket 函数关闭该套接字，释放该套接字相关的资源，具体代码如例 8-13 所示。

【例 8-13】

```
CChatDlg::~CChatDlg()
{
        if(m_socket)
```

```
                closesocket(m_socket);
    }
```

下面介绍利用主机名实现网络的访问。

上述 Chat 程序是通过输入对方的 IP 地址来向对方发送数据的，但是 IP 地址记忆起来不太方便，用户可能希望能够通过指定对方的主机名来发送数据。但是需要注意，在填充 SOCKADDR_IN 地址结构中的 sin_add.S_un.S_addr 成员时，需要使用 IP 地址，所以程序中需要将主机名转换为 IP 地址，这可以通过调用 gethostbyname 函数完成。该函数的原型声明如下。

```
    Struct hostent FAR * gethostbyname ( const char FAR * name );
```

Gethostbyname 函数从主机数据库中获取与主机名相对应的 IP 地址，该函数只有一个参数，是一个指向空终止的字符串。gethostbyname 函数返回 hostent 结构体类型的指针，该结构体类型的定义如下。

```
    Struct hostent{
        char FAR *       h_name,
        char FAR * FAR *         h_aliases,
        short                h_addrtype,
        short                h_length,
        char FAR * FAR *         h_addr_list
    };
```

hostent 结构体中最后一个成员 h_addr_list 是一个指针数组，它的每一个元素存放的是一个以网络字节顺序表示的主机 IP 地址，其中 h_addr_list[0]被定义为 h_addr 宏，这主要是为了兼容以前的软件。因为一台主机可能会有多个 IP 地址，利用主机名查询该主机的 IP 地址时，可能会返回多个 IP 地址，所以需要一个指针数组来存放这些地址。读者只需要记住，这个指针数组中的每个元素都是一个字符指针，其所指向的内存中存放的数据是以网络字节顺序存放的 IP 地址即可。

下面，首先在 Chat 程序的对话框中增加一个编辑框，允许用户在其中输入对方的主机名，并将其 ID 设置为 IDC_EDIT_HOSTNAME。

然后，在例 8-11 所示代码中的 OnBtnSend 函数中第 7 行代码后添加代码，定义一个 CString 对象类型的对象 strHostName，用来保存用户输入的主机名；再定义一个 HOSTENT 结构类型的指针，以便 gethostname 函数使用。

```
    CString strHostName;
    HOSTENT* pHost;
```

修改例 8-11 所示 OnBtnSend 函数中对 addrTo 变量的 sin_add.S_un.S_addr 成员进行赋值的代码（即用例 8-14 所示代码替换例 8-11 所示 OnBtnSend 函数中的第 8 行和第 9 行代码）。

【例 8-14】

```
    if(GetDlgItemText(IDC_EDIT_HOSTNAME,strHostName),strHostName=="")
    {
        ((CIPAddressCtrl*)GetDlgItem(IDC_IPADDRESS1))->GetAddress(dwIP);
        addrTo.sin_addr.S_un.S_addr=htonl(dwIP);
    }
    else
```

```
    {
        pHost=gethostbyname(strHostName);
        addrTo.sin_addr.S_un.S_addr=*((DWORD*)pHost->h_addr_list[0]);
    }
```

这时，首先调用 GetDlgItemText 函数获得主机名编辑框上的数据，然后判断得到的主机名是否为空，如果为空，则像先前一样，从 IP 地址控制上得到 IP 的地址，然后给地址结构体变量 addrTo 中的地址成员赋值；如果用户输入了主机名，则利用 gethostbyname 函数根据主机名获得 IP 地址，之后即可对地址结构体变量 addrTo 中的地址成员(sin_add.S_un.S_addr)赋值了。但这个成员需要的是网络字节顺序表示的 u_long 类型的地址，虽然 gethostbyname 函数返回的 IP 地址也是以网络字节顺序表示的，但其类型是 char*，所以这里先从地址数组(pHost->h_addr_list)中将得到的 IP 地址取出来，可能有多个 IP 地址信息，这里使用第一个地址即可，该元素是 char*类型，并且该指针所指向的内存中存放的是以网络字节顺序表示的 IP 地址。那么如何将一个 char*类型的数据转换为 u_long 类型呢？我们知道，只要内存中存放的数据类型是兼容的，那么各种指针之间可以相互转换。所以，可以先将 char*转换 DWORD*类型，也就是 u_long*类型，然后就可以利用取值符(*)取出该指针所指向的内存中的数据了。对 u_long 类型来说，其值占 4 个字节，因此时读取一个 u_long 类型的值，也就是取出了 4 个字节的数据。因为该内存中存放的是网络字节顺序的 IP 地址，所以这时取出的这 4 个字节的数值正好是 u_long 类型的 IP 地址。

运行 Chat 程序，在主机名编辑框中输入对方机器的主机名，并在发送数据编辑框中输入数据，然后按 Enter 键，即可在接收编辑框中看到接收到的数据。程序界面如图 8-5 所示。

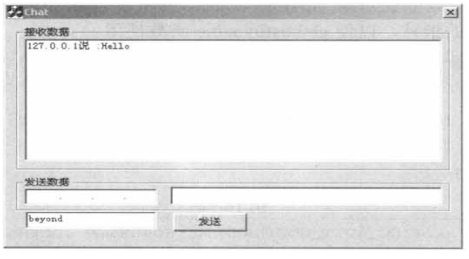


图 8-5　通过指定对方的主机名发送数据

读者可以看到，在接收编辑框中，显示的仍是发送方的 IP 地址，如果希望显示发送方的主机名，也就是说，在接收到对方发送的数据之后，需要将对方的 IP 地址转换为对方的主机名，则可以通过函数 gethostbyaddr 来完成，该函数的原型声明如下。

```
struct HOSTENT FAR * gethostbyaddr(
    const char FAR * addr,
    int len,
    int type
);
```

gethostbyaddr 函数有 3 个参数，第 1 个参数是 const char*类型，是一个指向以网络字节顺序表示的地址的指针；第 2 个参数是地址长度，对于 AF_INET 地址簇，地址长度必须是 4 个字节；第 3 个参数是地址类型，Windows 平台下必须设置为 AF_INET。

gethostbyaddr 函数将根据指定的 IP 地址，获取相应的主机信息。其返回值是一个 HOSTENT 结构体类型的指针。如果能找到相应的主机信息，那么返回的 HOSTENT 结构体的 h_name 数据成员就是主机名。

因此，在例 8-10 所示代码接收数据的消息响应函数 OnSock 中，定义一个 HOSTENT 结构体指针变量 pHost，并在调用 WSARecvFrom 函数接收数据之后，调用 gethostbyaddr 函数，修改进行数据格式化的代码，具体代码如下（即用下述代码段替换例 8-10 所示 OnSock 函数中第 19 行代码）。

```
HOSTENT      *pHost;
pHost=gethostbyaddr((char*)&addrFrom.sin_add.S_un.S_addr,4,AF_INET);
str.Format("%s 说：%s",pHost->h_name,wsabuf.buf);
// str.Format("%s 说：%s",inet_ntoa(addrFrom.sin_addr),wsabuf.buf);
```

由于 gethostbyaddr 函数需要一个以网络字节顺序表示的地址，从 SOCKADDR_IN 地址结构体变量 addrFrom 的 sin_add.S_un.S_addr 成员中可以取出一个 u_long 类型的，以网络字节顺序表示的 IP 地址数据，然而这里需要的是 const char*类型。刚才已经提到，只要我们清楚地知道数据的内存模型，就可以很容易地完成不同类型之间的转换。这里，addFrom.sin_add.S_un.S_addr 成员是 u_long 类型，需要的类型是 const char*，可以先对成员进行取地址操作，得到一个 u_long*的数据，再利用(char*)操作符将该结果强制转换即可。

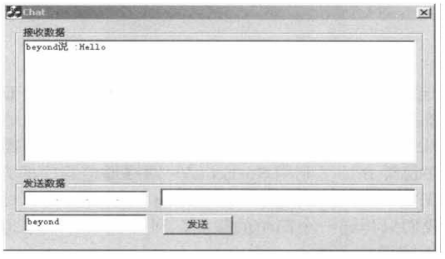


图 8-6　接收方显示数据方的主机名

在得到对方主机的信息后，通过 HOSTENT 结构体变量 pHost 的 h_name 成员取出主机名称，并进行相应的格式化。

再次运行 Chat 程序，在主机名编辑框中输入对方的主机名，并在发送数据编辑框中输入要发送的数据，按 Enter 键，即可在接收编辑框中看到接收到的数据。程序界面如图 8-6 所示。可以看到，这时显示的是发送数据方的主机名。也可以在 IP 地址控件中输入 IP 地址，然后发送数据，同样，在接收数据编辑框中显示的也是发送数据方的主机名。

8.2　进程间通信

当一个进程启动后，操作系统为其分配 4GB 的私有地址空间。位于同一个进程中的多个线程共享一个地址空间，因此线程之间的通信非常简单。然而，由于每个进程所拥有的 4GB 地址空间都是私有的，一个进程不能访问另一个进程地址空间中的数据，因此进程间的通信相对困难。在 Windows 平台下，系统为我们提供了多种进程通信的机制，前面已介绍了 Socket 编写网络通信的方法。实际上，网络编程就是两个进程，或多个进程间的通信，但 Socket 编程需要我们对网络协议有所了解。此外，虽只传递一个简单的数据，但利用 Socket 编程也需要较多的编码。

8.2.1　匿名管道

1．基础概念

匿名管道是一个未命名的、单向管道，通常用来在一个父进程和一个子进程之间传输数据。匿名管道只能实现本机机器上两个进程间的通信，而不能实现跨网络的通信。

为了创建匿名管道，需要调用 CreatePipe 函数。该函数的原型声明如下。

```
BOOL CreatePipe(
    PHANDLE hReadPipe，
    PHANDLE hWritePipe，
    LPSECURITY_ATTRIBUTES lpPipeAttibutes,
    DWORD nSize
);
```

CreatePipe 函数将创建一个匿名管道，返回该匿名管道的读写句柄。该函数有 4 个参数，其含义分别如下。

1）hReadPipe 和 hWritePipe：这两个参数都是 out 类型，即作为返回值使用。前者返回管道的读取句柄，后者接收管道的写入句柄。也就是说，在程序中需要定义两个句柄变量，将它们的地址分别传递给这两个参数，然后 CreatePipe 函数将通过这两个参数返回创建的匿名管道的读写句柄。

2）lpPipeAttibutes：一个指向 SECURITY_ATTRIBUTES 结构体的指针，检测返回的句柄是否被子进程继承。如果此进程为 NULL，则句柄不能被继承。在前面，凡是需要 SECURITY_ATTRIBUTES 结构体指针的地方，我们传递的都是 NULL 值，让系统为创建的对象赋予默认的安全描述符，而函数所返回的句柄将不能被子进程继承。但在本节匿名管道的例子中，不能再为此参数传递 NULL 值，因为匿名管道只能在父子进程之间进行通信。子进程如果想要获得匿名管道的句柄，只能从父进程继承而来。当一个子进程从其父进程继承了匿名管道的句柄后，这两个进程就可以通过该句柄进行通信了。所以，在本节匿名通道的例子中，必须构造一个 SECURITY_ATTRIBUTES 结构体变量，结构体如下。

```
typedef struct _SECURITY_ATTRIBUTES{
    DWORD nLength;
    LPVOID lpSecurityDescriptor;
    BOOL bInheritHandle;
}SECURITY_ATTRIBUTES, *PSECURITY_ATTRIBUTES;
```

SECURITY_ATTRIBUTES 结构体有 3 个成员，第 1 成员（nLength）指定该结构体的大小；第 2 个成员（lpSecurityDescriptor）是一个指向安全描述符的指针，在本节匿名管道的例子中，可以给这个成员传递 NULL 值，使系统为创建的匿名管道赋予默认的安全描述符；第 3 个成员（bInheritHandle）很关键，该成员指定所返回的句柄能否被一个新的进程继承，如果此成员为 TRUE，那么返回的句柄能够被新进程继承。在本节匿名管道的例子中，需要将成员设置为

TRUE，使子进程可以继承父进程创建的匿名管道的读写句柄。

3）nSize：指定管道的缓存大小，该大小仅仅是一个建议值，系统将使用这个值来计算适当的缓冲区大小。如果此参数是 0，则系统使用默认的缓冲区大小。

2．进程创建

为了启动一个进程，可以调用 CreateProcess 函数。该函数的原型声明如下。

```
BOOL CreateProcess(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandle,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStrartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

其参数含义分别如下。

1）lpApplicationName：一个指向 NULL 终止的字符串，用来指定可执行程序的名称。该名称可以是该程序的完整路径和文件名，也可以是部分名称。如果是后者，则 CreateProcess 函数在当前路径下搜索可执行文件名，但不会使用搜索路径进行搜索。注意，一定要加上扩展名，系统不会自动假设文件名有.exe 扩展名。

lpApplicationName 参数可以为 NULL，这时，文件名必须是 lpCommandLine 指向的字符串中的第一个空格界定的标记。如果使用了包含空格的长文件名，那么应该使用引导将该名称包含起来，以表明文件名的结束和参数的开始。否则文件名会产生歧义。例如，“c:\program files\sub dir\program name”字符串会被解释为多种形式，系统将按照下面的顺序来进行处理：

```
c:\program files\sub dir\program name
c:\program files\sub.exe dir\program name
c:\program files\sub dir\program.exe name
c:\program files\sub dir\program name.exe
```

2）lpCommandLine：一个指向 NULL 终止的字符串，用来指定传递给新进程的命令行字符串。系统会在该字符串的最后增加一个 NULL 字符，如果有必要，它会去掉首尾空格。

可以在 lpApplicationName 参数中传递可执行文件的名称，在 lpCommandLine 参数中传递命令行的参数。但应注意，如果在 lpCommandLine 参数中传递了一个可执行的文件名，并且没有包含路径，那么这时 CreateProcess 函数将按照以下顺序搜索可执行文件。

① 应用程序被装载的目录。

② 父进程的目录。

③ Windows Me/98/95：Windows 系统目录。Windows NT/2000：32 位 Windows 系统目录，即 C:\WINNT\System32 目录。

④ Windows NT/2000：16 位。Windows 系统目录：C:\WINNT\System32 目录。

⑤ Windows 目录。

⑥ PATH 环境变量中列出的目录。

可以将文件名和命令行参数构造为一个字符串，一并传递给这个参数，当 CreateProcess 函数分析 lpCommandLine 参数所指向的字符串时，它将查看该字符串中以空格分隔的第一标记，并加上该标记，即可得到将要运行的可执行文件的名称。如果这个可执行文件的文件名没有扩展名，则假设它的扩展名为.exe，当然，如果文件名包含全路径，那么系统将使用全路径查看可执行文件，并且不再搜索上述目录。

lpCommandLine 参数也可以为空，这时，CreateProcess 函数将使用 lpApplicationName 参数作为命令行。这两个参数的区别在于，如果在 lpApplicationName 参数中指定可执行文件名，那么系统将只在当前目录下查找该文件，如果没有找到，则返回失败。另外，系统不会为该文件加上扩展名；在 lpCommandLine 参数中指定可执行文件名，没有找到该文件，CreateProcess 函数才会失败返回。另外，如果在 lpCommandLine 参数指定文件名时没有加扩展名，那么这个函数会自动添加.exe 扩展名。通常在调用 CreateProcess 函数时，将可执行文件名和命令行参数都传递给 lpCommandLine 参数。

3）lpProcessAttributes 和 lpThreadAttributes：都是指向 SECURITY_ATTRIBUTES 结构体的指针。当调用 CreateProcess 函数创建新进程时，系统将为新进程创建一个进程内核对象和一个线程内核对象，后者用于进程的主线程。而 lpProcessAttributes 和 lpThreadAttributes 两个参数就分别用来设置新进程的进程对象和线程对象的安全性，以及指定父进程将来创建的其他子进程是否可以继承这两个对象的句柄，在程序中不需要创建其他子进程，可以为这两个参数传递 NULL，使系统为这两个对象赋予默认的安全性描述符。

4）bInheritHandle：用来指定父进程随后创建的子进程是否能够继承父进程的对象句柄。如果该参数为 TRUE，那么父进程的每个继承的打开句柄都能被子进程继承。继承的句柄与原始句柄拥有同样的值和访问特权。在下面的例子中，将把这个参数设置为 TRUE，让子进程继承父进程创建的管理的读写句柄。

5）dwCreationFlags：指定控制优先级类和进程创建的附加标记。如果只是为了启动子进程，并不需要设置它创建的标记，则可以直接将此参数设置为 0。

6）lpEnvironment：一个指向环境块的指针，如果参数是 NULL，那么新进程使用调用进程的环境。通常都是给此参数传递 NULL。

7）lpCurrentDirectory：一个指向空终止的字符串，用来指定子进程当前的路径，这个字符串必须是一个完整的路径名，包括驱动器的标识符，如果此参数为 NULL，那么新的子进程将与调用进程，即父进程拥有同样的驱动器和目录。

8）lpStrartupInfo：一个指向 STARUPINFO 结构体的指针，用来指定新进程的主窗口如何显示。该结构体的定义如下。

```
typedef struct _STARUPINFO{
    DWORD       cb;
    LPTSTR      lpReserved;
    LPTSTR      lpDesktop;
```

```
    LPTSTR      lpTitle;
    DWORD       dwX;
    DWORD       dwY;
    DWORD       dwXSize;
    DWORD       dwYSize;
    DWORD       dwXCountChars;
    DWORD       dwYCountChars;
    DWORD       dwFillAttibute;
    DWORD       dwFlags;
    WORD        wShowWindow;
    WORD        cbReserved2;
    LPBYTE      lpReserved2;
    HANDLE      hStdInput;
    HANDLE      hStdOutput;
    HANDLE      hStdError;
}STARUPINFO, *LP STARUPINFO;
```

可以看到，STARUPINFO 结构体的成员比较多，对这种拥有很多成员的结构体，在使用时并不需要为其所有成员都赋值，那么如何才能知道哪些成员能满足我们的需要呢？这里为读者介绍一个技巧，这种结构体，开始时可以大概浏览一下，找到一些特殊成员，如上述的 dwFlags 成员，通过其字面意思，猜测该字段可能用来设置一个标记。一个结构体中的标记往往就是用来设置在什么情况下应该使用该结构中的哪些成员。对这里的 dwFlags 成员来说，如果选择 STARTF_USESTDHANDLES 标记，那么将使用标准输入、标准输出和标准错误句柄，也就是说，这时只需要为 STARUPINFO 结构体中的这 3 个成员赋值即可。另外，在很多结构体中都有类似于 cb 或 nlens 的成员，它们都用来表示该结构体本身的大小，以字节为单位，通常都需要为此类成员赋值，否则函数调用可能会失败。因此，在使用 STARUPINFO 结构体时，还应设置其 cb 成员的值，即该结构体的大小。

9）lpProcessInformation：作为返回值使用，是一个指向 LPPROCESS_INFORMATION 结构体的指针，用来接收关于新进程的标识信息。该结构体的定义如下。

```
Typedef struct _ LPPROCESS_INFORMATION{
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} LPPROCESS_INFORMATION;
```

LPPROCESS_INFORMATION 结构体有 4 个成员。前两个成员是 hProcess 和 hThread，分别用于标识新创建的进程句柄和新创建进程的主线程句柄；后两个成员是 dwProcessId 和 dwThreadId，分别是全局进程标识符和全局线程标识符，前者可以用来标识一个进程，后者可以用来标识一个线程。

当启动一个线程时，系统为会此进程分配一个标识符，同时这个进程中的线程也会被分配一个标识符，在一个进程运行时，该进程的标识符和线程的标识符是唯一的，但应注意，当该进程停止运行时，该进程的标识符和其线程的标识符可能会被系统分配给另一个进程和另一个

线程使用。如果一个函数调用依赖于进程的标识符或者线程的标识符，那么就要确保该进程当前处于运行状态，否则结果无法预料。

3．父进程的实现

下面，利用匿名管道实现进程间的通信。首先实现父进程，新建一个单文档类型的 MFC 应用程序，工程取名为 Parent。为该工程增加一个子菜单，名称为“匿名管道”。再为该子菜单添加 3 个菜单项，并分别为它们添加相应的命令响应函数，本例中使用 CParentView 类接收这些命令响应函数。各菜单项的 ID、名称及响应函数如表 8-3 所示。

表 8-3　添加的菜单项及响应函数

ID	菜单名称	响应函数
IDM_PIPE_CREATE	创建管道	OnPipeCreate
IDM_PIPE_READ	读取数据	OnPipeRead
IDM_PIPE_WRITE	写入数据	OnPipeWrite

为 CParentView 类增加以下两个私有成员变量，即两个句柄，它们将分别作为匿名管道的读写句柄使用。

```
private：
    HANDLE hWrite;
    HANDLE hRead;
```

在 CParentView 类的构造函数中将它们初始化为 NULL。

```
CParentView：：CParentView()
{
    //TODO: add construction code here
    hRead=NULL;
    hWrite=NULL;
}
```

在 CParentView 类的析构函数中，如果判断出这两个变量有值，则调用 CloseHandle 函数关闭这两个变量。

```
CParentView：：~CParentView()
{
    if(hRead)
        CloseHandle(hRead);
    If(hWrite)
        CloseHandle(hWrite);
}
```

（1）创建匿名管道

在“创建管道”菜单项响应函数 OnPipeCreate 中调用 CreatePipe 创建匿名管道，返回管道的读写句柄。代码如例 8-15 所示。

【例 8-15】

```
void CParentView::OnPipeCreate()
{
  // TODO: Add your command handler code here
  SECURITY_ATTRIBUTES sa;
  sa.bInheritHandle=TRUE;
  sa.lpSecurityDescriptor=NULL;
  sa.nLength=sizeof(SECURITY_ATTRIBUTES);
  if(!CreatePipe(&hRead,&hWrite,&sa,0))
  {
        MessageBox("创建匿名管道失败！");
        return;
  }
  STARTUPINFO sui;
  PROCESS_INFORMATION pi;
  ZeroMemory(&sui,sizeof(STARTUPINFO));
  sui.cb=sizeof(STARTUPINFO);
  sui.dwFlags=STARTF_USESTDHANDLES;
  sui.hStdInput=hRead;
  sui.hStdOutput=hWrite;
  sui.hStdError=GetStdHandle(STD_ERROR_HANDLE);

  if(!CreateProcess("..\\Child\\Debug\\Child.exe",NULL,NULL,NULL,
            TRUE,0,NULL,NULL,&sui,&pi))
  {
        CloseHandle(hRead);
        CloseHandle(hWrite);
        hRead=NULL;
        hWrite=NULL;
        MessageBox("创建子进程失败！");
        return;
  }
  else
  {
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
  }
}
```

在例 8-17 所示代码中，首先定义了一个安全结构体(SECURITY_ATTRIBUTES)类型的变量 sa，并对其成员分别赋值。这里需要将 bInheritHandle 成员设置为 TRUE，让子进程可以继承父进程创建的匿名管道的读写句柄；将安全描述符成员(lpSecurityDescriptor)设置为 NULL，让系统为创建的匿名管道赋予默认的安全描述符；长度成员(nLength)可以利用 sizeof 函数得到 SECURITY_ATTRIBUTES 结构体的长度。

然后，调用 CreatePipe 函数创建匿名管道(第 5 行代码)，前两个参数是返回的管道的读取

和写入句柄，第 3 个参数是刚刚定义的安全结构的地址，最后一个参数设置为 0，让系统使用默认的缓冲区大小。CreatePipe 函数如果调用失败，则返回一个 0 值，这时提示用户 “创建匿名管道失败!”，并立即返回。

如果创建匿名管道成功，则启动子进程，并将匿名管道的读写句柄传递给子进程。为了启动一个进程，可以调用 CreateProcess 函数，根据前面的介绍，我们知道函数的第 9 个参数需要一个 STARTUPIOFO 结构体类型的值，因此上述代码中定义了一个这种结构体类型的变量 sui（第 10 行代码），因为该结构体中有多个成员，而我们只能用到了其中的一小部分，在没有将其他成员置为 0 之前，它们都只是一些随机的值，如果将这些随机的值传递给 CreateProcess 函数，则可能会影响该函数执行的结构，所以首先应该调用 ZeroMemory 函数将 sui 变量中所有成员设置为 0（第 12 行代码）。

然后，为 sui 变量的几个必要成员赋值，因此，这里首先用 STARTUPIOFO 结构体长度为 sui 变量的 cb 成员赋值；再设定标志成员 dwFlags 的值为 STARTF_USESTDHANDLES，当采用此标志时，表示当前 STARTUPIOFO 结构体变量中的标准输入、标准输出和标准错误句柄成员是有用的，本例将子进程的标准输入句柄、输出句柄设置为管道的读、写句柄。当调用 CreateProcess 函数启动一个子进程时，它将继承父进程中所有可继承的已打开的句柄，但在子程序中无法知道它所继承的句柄中哪一个是管道的读句柄，哪一个是管道的写句柄，因为管道的读写句柄并不是通过参数，或者其他方式传递进来的，它们只是子进程从其父进程中继承的众多句柄中的两个。为了让子进程从众多继承的句柄中区分出管道的读、写句柄，必须将子程序的特殊句柄设置为管道的读写句柄，因此这里将子进程的标准输入和输出句柄分别设置为管道的读、写句柄。设置子进程的标准错误设备句柄，这可以通过 GetStdHandle 函数得到，该函数可以获得标准输入、标准输出，或者一个标准错误输出句柄。该函数的原型声明如下。

```
HANDLE GetStdHandle( DWORD nStdHandle);
```

GetStdHandle 函数有一个 DWORD 类型的参数，通过为该参数指定不同的值来得到不同的句柄。该参数的取值如表 8-4 所示。

表 8-4　nStdHandle 参数取值

值	含　义
STD_INPUT_HANDLE	标准输入设备句柄
STD_OUTPUT_HANDLE	标准输出设备句柄
STD_ERROR_HANDLE	标准错误设备句柄

由表 8.5 可知，如果将 GetStdHandle 函数的参数指定为 STD_INPUT_HANDLE，那么该函数返回的是一个标准的错误设备句柄。应注意，这里得到的是父进程的标准错误句柄，也就是说，将子进程的标准错误句柄设置为父进程的标准错误句柄。虽然这个标准错误设备句柄在本程序中没有被使用，但读者应该清楚这里得到的是父进程的错误句柄。

上例所示的 OnPipeCreate 函数调用了 CreateProcess 函数创建子进程（第 9 行代码）。将该函数的第一个参数设置为子进程的应用程序的文件名，这里先指定为“..\\Child\\Debug\\Child.exe”。下面编写 Child 子程序，并把这个子程序的目录与本程序所在目录保持平级；第 2 个参数指定命令行参数，本例不需要指定，所以设置为 NULL；第 3 个参数和第 4 个参数分别

是进程安全属性和线程安全属性，均设置为 NULL，使用默认的安全属性；第 5 个参数设置为 TRUE，让父进程的每个可继承的打开句柄都能被子进程继承；第 6 个参数是创建标志，本例并不需要指定，所以设置为 0；第 7 个参数设置为 NULL，让新进程使用调用进程的环境；第 8 个参数把当前路径设置为 NULL，让子进程与父继承拥有同样的驱动器和路径；第 9 个参数是 PROCESS_INFORMATION 结构体指针，同样，先定义一个该结构体类型的变量 pi（第 11 行代码），然后将变量的地址传递给这个参数。

程序需要对 CreateProcess 函数的返回值进行判断，如果该函数调用失败，则返回 0 值，这时可调用两次 CloseHandle 函数关闭管道的读、写句柄，并将这两个句柄设置为 NULL（为了避免在 CParentView 类的析构函数再次调用 CloseHandle 函数而关闭这些句柄，将它们设置为 NULL），然后提示用户 “创建子进程失败!”并返回；如果 CreateProcess 函数调用成功，则调用 CloseHandle 函数关闭多返回的子程序的句柄和子程序中主线程的句柄。

为什么要这么做呢？前面已经提到，在创建一个新进程时，系统会为该系统建立一个进程内核对象和一个线程内核对象，而内核对象都有一个使用计数，系统会为这两个对象赋予初始的使用计数 1，在 CreateProcess 函数返回之前，它将打开创建的进程对象和线程对象，并将每个对象与进程和线程相关的句柄放在最后一个参数 PROCESS_ INFORMATION 结构体变量的相应成员中。当 CreateProcess 函数在其内部打开这些对象时，每个对象的使用计数变为 2，如果在父进程中不需要使用子进程的这两个句柄，则可以调用 CloseHandle 函数关闭它们，系统会将子进程的进程内核对象和线程内核对象的使用计数减 1，当子进程终止运行时，系统会再将这些使用计数减 1，这时子进程的进程内核对象和线程内核对象的计数都变为 0，这两个内核对象即能够被释放。所以在编程中，不需要这些内核对象时，应该调用 CloseHandle 函数关闭它们。

以上就是在父进程中创建匿名管道的实现代码。当子进程启动后，父子进程间即可通过创建的匿名管道来读取数据和写入数据。对于管道的读取和写入实际上是通过调用 ReadFile 和 WriteFile 这两个函数完成的。前面已经提到过，这两个函数不仅能够完成对文件的读写，还可以完成如控制台和管道这两类对象的读写操作。

（2）读取数据

下面代码作用是在 OnPipeRead 函数中实现匿名管道的读取操作，结果如例 8-16 所示。

【例 8-16】

```
void CParentView::OnPipeRead()
{
        // TODO: Add your command handler code here
        char buf[100];
        DWORD dwRead;
        if(!ReadFile(hRead,buf,100,&dwRead,NULL))
        {
              MessageBox("读取数据失败！");
              return;
         }
         MessageBox(buf);
}
```

在实现读取操作时，首先定义了一个 char 类型的字符数组 buf，用来存放将要读取到的数据。再定义一个 DWORD 类型的变量 dwRead，用来保存实际读取的字节数。调用 ReadFile 函数，利用匿名管道的读句柄从管道中读取数据。该函数如果调用失败，则返回 0 值，这时提示用户“读取数据失败！”；如果 ReadFile 函数调用成功，则调用 MessageBox 函数显示读取到的数据。

（3）写入数据

下面代码作用是在 OnPipeWrite 函数中实现匿名管道的写入操作，结果如例 8-17 所示。

【例 8-17】

```
void CParentView::OnPipeWrite()
{
    // TODO: Add your command handler code here
    char buf[]="www.baidu.com ";
    DWORD dwWrite;
    if(!WriteFile(hWrite,buf,strlen(buf)+1,&dwWrite,NULL))
    {
        MessageBox("写入数据失败！");
        return;
    }
}
```

同样的，在实现写入操作时，首先定义了一个 char 类型的字符数组 buf，其内容是一个网址，即“http://www.baidu.com”，就是我们将要写入的数据。然后，定义了一个 DWORD 类型的变量 dwWrite，用来保存实际写入字节数。调用 WriteFile 函数利用匿名管道的写入句柄管道写入数据。该函数如果调用失败，则返回 0 值，这时提示用户“写入数据失败!”，然后调用 return 语句返回。

最后，需要利用 Build 命令生成 Parent 程序。

以上就是利用匿名管道实现父子进程间通信的父进程程序的实现，在父进程中创建匿名管道，返回该管道的读、写句柄，然后调用 CreateProcess 函数启动子进程，并且将子进程的标准输入和标准输出句柄设置为匿名管道的读、写句柄，相当于给该管道的读写句柄做上了标记，传递给子进程，在子进程中得到自己的标准输入和标准输出句柄时，相当于得到了匿名管道的读、写句柄。

4．子进程的实现

下面开始编写利用匿名管道实现父子进程间通信的子进程程序，同样，新建一个单文档类型的 MFC 应用程序，工程取名为 Child，并将该工程添加到上面 Parent 程序所在的工作区中，同时设置该工程所在的目录与上述 Parent 工程的目录平级。

为 Child 工程增加一个子菜单，名称为“匿名管道”。再为该子菜单添加两个菜单项，并分别为它们添加相应的响应函数，本例选择 CChildView 类接收这些命令响应函数。各菜单项的 ID、名称及响应函数如表 8-5 所示。

表 8-5　添加的菜单项及响应函数

ID	菜单名称	响应函数
IDM_PIPE_READ	读取数据	OnPipeRead
IDM_PIPE_WRITE	写入数据	OnPipeWrite

同样，为 CChildView 类增加以下两个私有成员变量，即两个句柄，它们将分别作为匿名管道的读取和写入句柄。

```
private:
    HANDLE hWrite;
    HANDLE hRead;
```

在 CChildView 类的构造函数中将它们都初始化为 NULL。

```
CChildView::CChildView()
{
    // TODO: add construction code here
    hRead=NULL;
    hWrite=NULL;
}
```

在 CChildView 类的析构函数中，如果判断这两个变量有值，则调用 CloseHandle 函数关闭这两个变量。

```
CChildView::CChildView()
{
    if(hRead)
        CloseHandle(hRead);
    If(hWrite)
        CloseHandle(hWrite);
}
```

（1）获取管道的读取和写入句柄

为了利用父进程创建的匿名管道进行通信，在子进程中，首先要得到子进程的标准输入和标准输出句柄，这可以在 CChildView 类窗口完全创建成功后获取。因此，根据前面的知识，可以为 CChildView 类增加虚函数 OnInitialUpdate，这个函数是当窗口成功创建之后，第一个调用的函数。在此函数中可以通过调用 GetStdHandle 函数获取子进程的标准输入和标准输出句柄，具体代码如例 8-18 所示。

【例 8-18】

```
void CChildView:: OnInitialUpdate()
{
    CView:: OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
    hRead=GetStdHandle(STD_INPUT_HANDLE);
    hWrite=GetStdHandle(STD_OUTPUT_HANDLE);
}
```

（2）读取数据

得到匿名管道的读取和写入句柄后，即可利用这些管道句柄读取和写入数据。应该在 OnPipeRead 函数中实现从父进程创建的管道上读取数据。子进程中读取数据的实现与在 Parent 程序中读取数据的实现是一样的，具体实现代码如例 8-19 所示。

【例 8-19】

```
void CChildView:: OnPipeRead()
{
    // TODO: Add your command handler code here
    Char buf[100];
    DWORD dwRead;
    if(!ReadFile(hRead,buf,100,&dwRead,NULL)
    {
        MessageBox("读取数据失败");
        Returan;
}
MessageBox(buf);
}
```

（3）写入数据

在 OnPipeWrite 函数中，实现向父进程的管道中写入数据的功能，子进程写入数据的实现与 Parent 程序中写入数据的实现是一样的，但为了区分父子进程写入的数据，这里让子进程写入不同的数据，具体实现代码如例 8-20 所示。

【例 8-20】

```
void CChildView::OnPipeWrite()
{
    // TODO: Add your command handler code here
    Char buf[]="匿名管道测试程序";
    DWORD dwWrite;
    if(!WriteFile(hWrite,buf,strlen(len)+1,&dwWrite,NULL)
    {
        MessageBox("写入数据失败");
        Returan;
}
MessageBox(buf);
}
```

最后，利用 Build 命令生成 Child 程序。

以上就是利用匿名管道实现进程间通信的子进程程序的实现。在子进程中，只需要获得父进程创建的匿名管道的读、写句柄，然后利用这两个句柄实现从管道读取数据，或者向管道写入数据即可。

现在可以测试 Parent 进程和 Child 进程的通信效果。如果我们按照通常启动程序的方法，独立地启动了这两个进程，那么这时这两个进程间之间是否可以进行通信呢？读者应注意，对于匿名管道来说，它只能在父、子进程之间进行通信。两个进程如果想要具有父子关系，则必须由父进程通过调用 CreateProcess 函数来启动子进程。若单独启动这两个进程，则它们之间并

没有父子关系，并不会因为程序名一个是“Parent”，另一个是“Child”，它们之间就具有父子关系了。对于子进程来说，它必须由父进程来启动。所以这里应该先启动 Parent 程序，然后选择“创建管道”选项来启动子进程。可以在 Parent 程序中选择“写入数据”选项，即在父进程中向创建的匿名管道写入数据，然后在 Child 程序中选择“读取数据”选项，将会看到 Child 程序弹出一个消息框，提示接收到数据“www.baidu.com”，如图 8-7 所示。同样的，子进程也可以向匿名管道写入数据，在父进程中读取数据。在 Child 程序中选择“写入数据”选项，然后在 Parent 程序中选择“读取数据”选项，将会看到 Parent 程序弹出一个消息框，提示接收到数据 “匿名管道测试程序”，如图 8-8 所示。

另外，利用匿名管道还可以实现在同一个进程内读取和写入数据的功能。例如，可以在 Parent 进程中先通过选择“创建管道”选项创建匿名管道，接着选择“写入数据”选项，向匿名管道写入数据，选择“读取数据”选项，从该管道读取数据，将会看到程序的结果也是正确的。

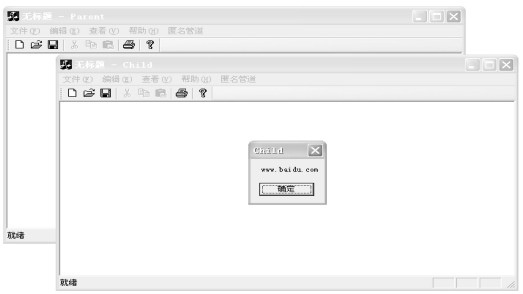


图 8-7　利用匿名管道实现父子进程间通信运行结果（一）

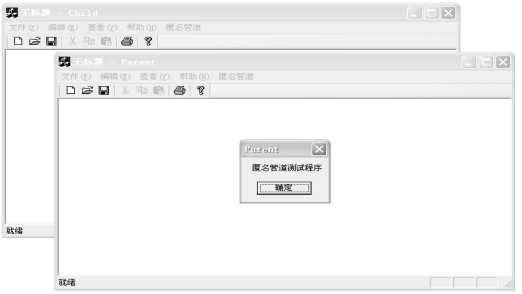


图 8-8　利用匿名管道实现父子进程间通信运行结果（二）

以上就是对匿名管道的使用，它主要用来在父子进程间进行通信。利用匿名管道实现父子进程间通信时，需要注意一点，因为匿名管道没有名称，所以只能在父进程中调用 CreateProcess 函数创建子进程，再将管道的读、写句柄传递给子进程。

8.2.2　命名管道

1．基础概念

命名管道通过网络来完成进程间的通信，它屏蔽了底层的网络协议细节。我们在不了解网络协议的情况下，也可以利用命名管道来实现进程间的通信。上面介绍的匿名管道只能在本地机器上的父子进程间进行通信，而命名管道不仅可以在本机上实现两个进程间的通信，还可以跨网络实现两个进程间的通信。

命名管道充分利用了 Windows NT 和 Windows 2000 内建的安全机制。在创建管道时，可以指定具体访问权限的用户，而其他用户不能访问这个管道。如果采用 Sockets 编写网络应用，那么为了完成用户身份验证需要程序员自行编码实现，而采用命名管道不需要再编写身份验证的代码。

将命名管道作为一种网络编程方案时，它实际上建立了一个客户机/服务器通信体制，并在其中可靠地传输数据。命名管道是围绕 Windows 文件系统设计的一种机制，采用“命名管道文件系统(Name Pipe File System，NPFS)”接口，因此，客户机和服务器可利用标准的 Win32 文件系统函数（如 ReadFile 和 WriteFile）来进行数据的收发。命名管道服务器和客户端的区别在于：服务器是唯一一个有权创建命名管道的进程，也只有它才能接收管道客户机的连接请求。而客户机只能同一个现成的命名管道服务器建立连接。命名管道服务器只能在 Windows NT 或 Windows 2000 上创建，因此，无法在两台 Windwos 95 或者 Windows 98 计算机之间利用管道进行通信。但客户机可以是 Windows 95 或 Windows 98 计算机，与 Windows NT 或 Windows 2000 计算机连接通信。

命名管道提供了两种基本通信模式：字节模式和消息模式。在字节模式下，数据以一个连续的字节流的形式在客户机和服务器之间流动。而在消息模式下，客户机和服务器通过一系列不连续的数据单位，进行数据的收发，每次在管道上发出了一条消息后，它必须作为一条完整的消息读入。

在程序中如果要创建一个命名管道，则需要调用 CreateNamedPipe 函数。该函数的原型声明如下。

```
HANDLE CreateNamedPipe(
    LPCTSTR      lpName,
    DWORD        dwOpenMode,
    DWORD        dwPipeMode,
    DWORD        nMaxInstances,
    DWORD        nOutBufferSize,
    DWORD        nInBufferSize，
    DWORD        nDefaultTimeOut，
    LPSECURITY_ATTRIBUTES      lpSecurityAttributes
);
```

CreateNamedPipe 函数创建一个命名管道的实例，并返回该命名管道的句柄。一个命名管道的服务器进程使用该函数创建命名管道的第一个实例，并建立它的基本属性，或者创建一个现有的命名管道的新实例。如果需要创建一个命名管道的多个实例，则需要多次调用 CreateNamedPipe 函数。该函数各个参数的含义分别如下。

1）dwOpenMode：指定管道的访问方式、重叠方式、写直通方式、管道句柄的安全访问方式。

这个参数的管道访问方式必须是表 8-6 所示值之一，并且管道的每一个实例都必须有同样的访问方式。

该参数还可以包含表 8-7 所示的标记中的一个或多个，用来指定写直通方式和重叠方式。

关于重叠操作，前面已经介绍了，如果采用了重叠操作，对管道的读写函数将立即返回。当该操作完成之后，系统会通过一种方式通知调用进程，本例将创建一个允许重叠操作的命名管道。

该参数还可以包含表 8-8 所示的标记的一个或多个，用来指定管道的安全访问方式。

表 8-6　命名管道的访问方式

管道访问方式	说　明
PIPE_ACCESS_DUPLEX	双向模式，服务器进程和客户端进程都可以从管道读取数据并向管道中写入数据。该模式等价于指定 GENERIC_READ|GENERIC_WRITE。当客户端调用 CreateFile 函数与管道连接时，可以指定 GENERIC_READ|GENERIC_WRITE，或者两者都指定
PIPE_ACCESS_INBOUND	管道中的数据流向只能从客户端到服务器进程，相当于指定 GENERIC_READ。也就是说，如果在服务器端创建命名管道时指定 PIPE_ACCESS_INBOUND 访问方式，那么服务器端只能读取数据，而客户端只能向管道写入数据
PIPE_ACCESS_OUTBOUND	管道中的数据流向只能是从服务器到客户端进程。服务器只能向管道写入数据，而客户端只能从管道读取数据

表 8-7　写直通和重叠方式

写直通和重叠方式	说　明
FILE_FLAG_WRITE_THROUGH	允许写直通方式。该方式只影响对字节类型管道的写入操作，并且只有当客户端与服务器端进程位于不同的计算机上时才有效。如果采用了该方式，那么只有等到要写入命名管道的数据通过网络传送过去，并且放在了远程计算机的管道缓冲区中后，写数据的函数才会成功返回。如果没有采用该方式，那么系统会通过缓冲数据来提高网络操作的效率，直到积累的字节数达到了最小值，或超过了最大时间值
FILE_FLAG_OVERLAPPED	允许采用重叠模式，如果采用了该模式，那么那些可能会需要一定的时间才能完成的读写操作会立即返回。在重叠模式下，前台线程可以执行其他操作，而耗费时间的操作可以在后台进行。例如，在重叠模式下，一个线程可以在管道的多个实例上同时处理输入和输出操作，或者在同一个管道句柄上同时执行读写操作。如果没有指定重叠模式，那么在管道句柄上执行的读取和写入操作只有在这些操作完成之后才能返回。ReadFileEx 和 WriteFileEx 函数只能在重叠模式下使用管道句柄，而 ReadFile、WriteFile、ConnectNamedPipe 和 TransactNamedPipe 函数可以重叠方式执行，也可以采用同步方式执行

表 8-8　安全访问方式

安全访问方式	说　明
WRITE_DAC	调用者对命名管道的任意访问控制列表(ACL)都可以进行写入访问
WRITE_OWNER	调用者对命名管道的所有者可以进行写入访问
ACCESS_SYSTEM_SECURATY	调用者对命名管道的安全访问控制列表(SACL)可以进行写入访问

2）dwPipeMode：指定管道句柄的类型、读取和等待方式。管道句柄的类型可以取表 8-9 所示值之一。

表 8-9　管道句柄的类型

值	说　明
PIPE_TYPE_BYTE	数据以字节流的形式写入管道，该方式不能在 PIPE_READMODE_MESSAGE 读方式下使用
PIPE_TYPE_MESSAGE	数据以消息流的形式写入管道，该方式在 PIPE_READMODE_MESSAGE 和 PIPE_READMODE_BYTE 读方式下都可使用

读者应注意，同一个命名管道的每一个实例必须具有相同的类型。如果该参数值为 0，那么默认是字节类型方式。也就是说，通过这个参数，可以指定创建的是字节模式，还是消息模式的管道，如果是 PIPE_TYPE_BYTE，则创建的是字节模式，不能和 PIPE_TYPE_MESSAGE 读模式一起使用。因为当把命名管道指定为消息模式时，系统发送消息时有一个定界符，当以消息读的模式读取时，通过该定界符可以读取一条完整的消息，但如果采用字节读方式读取，将忽略该定界符而直接读取数据。所以，对消息模式的命名管道来说，可以采用消息读，也可以采用字节读的方式读取数据。但是，对字节模式的命名管道来说，数据是一种字节流格式，没有定界符，因此当采用消息读的模式读取时，不知道应该读取多少字节的数据才合适。

管道句柄的读取方式可以是 8-10 所示值之一，同一管道的不同实例可以指定不同的读取方式。如果该值置为 0，则默认是字节读方式。

表 8-10　管道句柄的读取方式

管道句柄的读取方式	说　明
PIPE_TYPE_BYTE	以字节流的方式从管道读取数据。这种方式在 PIPE_TYPE_BYTE 和 PIPE_TYPE_MESSAGE 类型下均可使用
PIPE_TYPE_MESSAGE	以消息的方式从管道读取数据。该方式只能在 PIPE_TYPE_MESSAGE 类型下使用

管道句柄的等待方式可以是表 8-11 所示值之一，同一管道的不同实例可以有不同的等待方式。如果该值设置为 0，则默认是阻塞方式。

表 8-11　管道句柄的等待方式

管道句柄的等地方式	说　明
PIPE_WAIT	允许阻塞方式，在这种方式下，ReadFile、WriteFile 或 ConnectNamedPipe 函数必须等到读取到数据，或写入所有数据，或有一个客户连接到来才能返回
PIPE_NOWAIT	允许非阻塞方式，在这种方式下，ReadFile、WriteFile 或 ConnectNamedPipe 函数总是立即返回的

3）nMaxInstances：指定管道能够创建的实例的最大数目。该参数的取值范围从 1 到 PIPE_UNLIMITED_INSTANCES。如果是 PIPE_UNLIMITED_INSTANCES，那么可以创建的管道实例数目仅仅受限于系统可使用的资源。例如，如果将这个参数值设置为 5，则最多可以创建该命名管道的 5 个实例，所指的最大实例数目是否指对同一个命名连接到这个命名管道的实例上呢？实际上，这里所指的最大实例数目是指对同一个命名管道最多所能创建的实例数目。如果希望同时能够连接 5 个客户端，那么必须调用 5 次 CreateNamedPipe 函数创建 5 个命名管道实例，然后才能同时接收 5 个客户端连接请求的到来。对同一个命名管道的实例来说，在某一时刻，它只能和一个客户端进行通信。

4）nOutBufferSize：指定为输出缓冲区所保留的字节数。

5）nInBufferSize：指定输入缓冲区的大小是可变的，保留给命名管道的每一端的实际缓冲区大小既可以是默认值，也可以是系统最小值、系统最大值，或延伸到下一个分配边界的一个指定值。

6）nDefaultTimeOut：指定默认的超时值，单位是 ms。同一个管道的不同实例必须指定同

样的超时值。

7）lpSecurityAttributes：指向 SECURITY_ATTRIBUTES 结构的指针，该结构指定了命名管道的安全描述符，并确定子进程是否可以继承这个函数返回的管道句柄。可以将这个参数设置为 NULL，让命名管道具有默认的安全描述符，而且该句柄不能被继承。

2．服务端程序

下面，利用命名管道实现进程间的通信。首先实现服务器端程序，新建一个单文档类型的 MFC 应用程序，工程取名为 NamedPipeSrv。再为该工程增加一个子菜单，名称为“命名管道”。为该子菜单添加 3 个菜单项，并分别为它们添加相应的响应函数，本例选择 CNamePipeSrvView 类接收这些响应函数。各菜单项的 ID、名称及响应函数如表 8-12 所示。

表 8-12　添加的菜单项及响应函数

ID	菜单名称	响应函数
IDM_PIPE_CREATE	创建管道	OnPipeCreate
IDM_PIPE_READ	读取数据	OnPipeRead
IDM_PIPE_WRITE	写入数据	OnPipeWrite

为 CNamedPipedSrvView 类增加一个句柄变量，用来保存创建的命名管道实例的句柄。

```
private:
    HANDLE hPipe;
```

在 CNamedPipeSrvView 类的构造函数中将其初始化为 NULL。

```
CNamedPipeSrvView：：CNamedPipeSrvView()
{
    //TODO: add construction code here
    hPipe=NULL;
}
```

在 CNamedPipeSrvView 类的析构函数中，如果判断该句柄有值，则调用 CloseHandle 函数关闭该句柄。

```
CNamedPipeSrvView：：~CNamedPipeSrvView()
{
    if(hPipe)
        CloseHandle(hPipe);
}
```

（1）创建命名管道

在 OnPipeCreate 函数中可以调用 CreateNamedPipe 函数创建命名管道。具体代码如例 8-21 所示。

【例 8-21】

```
void CNamedPipeSrvView::OnPipeCreate()
{
    // TODO: Add your command handler code here
    hPipe=CreateNamedPipe("\\\\.\\pipe\\MyPipe",
     PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED,
```

```
        0,1,1024,1024,0,NULL);
    if(INVALID_HANDLE_VALUE==hPipe)
    {
        MessageBox("创建命名管道失败！");
        hPipe=NULL;
        return;
    }
    HANDLE hEvent;
    hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);
    if(!hEvent)
    {
        MessageBox("创建事件对象失败！");
        CloseHandle(hPipe);
        hPipe=NULL;
        return;
    }
    OVERLAPPED ovlap;
    ZeroMemory(&ovlap,sizeof(OVERLAPPED));
    ovlap.hEvent=hEvent;
    if(!ConnectNamedPipe(hPipe,&ovlap))
    {
        if(ERROR_IO_PENDING!=GetLastError())
        {
            MessageBox("等待客户端连接失败！");
            CloseHandle(hPipe);
            CloseHandle(hEvent);
            hPipe=NULL;
            return;
        }
    }
    if(WAIT_FAILED==WaitForSingleObject(hEvent,INFINITE))
    {
        MessageBox("等待对象失败！");
        CloseHandle(hPipe);
        CloseHandle(hEvent);
        hPipe=NULL;
        return;
    }
    CloseHandle(hEvent);
}
```

利用 C 语言编程时，如果想要指定两个反斜杠，那么在代码中就需要输入四个反斜杠。所以，在例 8-22 所示代码中调用 CreateNamedPipe 函数时，将管道的名称指定为\\\\.\\pipe\\MyPipe；管道访问的模式设定为 PIPE_ACCESS_DUPLEX，即双向模式，服务器进程和客户端进程都可以从管道读取和向管道中写入数据。同时，指定 FILE_FLAG_OVERLAPPED 标志，允许重叠方式；第 3 个参数用来指定管道类型、读取和等待

方式，本例将其值设为 0，即默认为字节类型和字节读方式；第 4 个参数用来指定管道实例的最大数目，本例设置为 1，因为本程序是一个测试程序，只需要一个客户端连接即可；第 5 个和第 6 个参数分别用来指定输出缓冲区大小和输入缓冲区的大小，本例都设置为 1024；第 7 个参数指定超时值，本例设为 0；最后一个参数指定安全属性，本例设置为 NULL，让管道句柄使用默认的安全性。

如果 CreateNamedPipe 函数调用成功，则返回一个有效的管道句柄；否则返回 INVALID_HANDLE_VALUE，则可以调用 GetLastError 函数获得更多的错误信息。

读取数据在程序中可以对 CreateNamedPipe 函数的返回值进行判断，如果失败，则提示用户“创建命名失败！”，将管道句柄变量(hPipe)设置为 NULL，这样是为了避免程序失败时在 CNamedPipeSrvView 对象的析构函数中再次调用 CloseHandle 函数而关闭这个句柄，然后让 OnPipeCreate 函数直接返回；如果成功创建了命名管道的实例，则可以调用 ConnectNamedPipe 函数，等待客户端请求的到来。这个函数允许一个服务端进程等待一个客户端进程连接到一个命名管道的一个实例上。这个函数的命名不太好，让人感觉好像连接了服务器的命名管道，实际上这个函数的作用是让服务器等待客户端连接请求的到来。该函数声明如下。

```
BOOL   ConnectNamedPipe(HANDLE hNamedPipe, LPOVERLAPPED lpOverlapped);
```

ConnectNamePipe 函数有两个参数，其含义分别如下。

1）hNamedPipe：指向一个命名管道实例的服务器的句柄，该句柄由 CreateNamedPipe 函数返回。

2）lpOverlapped：指向一个 OVERLAPPED 结构的指针，如果 hNamedPipe 参数所标识的管道是用 FILE_FLAG_OVERLAPPED 标记打开的，则这个参数不能是 NULL，必须是一个有效的、指向一个 OVERLAPPED 结构的指针；否则该函数可能会执行错误。如果 hNamedPipe 参数所标识的管道是用 FILE_FLAG_OVERLAPPED 标记打开的，并且这个参数不是 NULL，则这个参数所指向的 OVERLAPPED 结构体中必须包含人工重置对象句柄。

于是，上述 OnPipeCreate 函数调用 CreateEvent 函数创建了一个人工重置事件对象句柄（注意，第 2 个参数值一定为 TRUE）。CreateEvent 函数如果调用失败，将返回 NULL，所以对该函数的返回值进行判断，如果调用失败，则提示用户“创建事件失败！”，并在调用 return 语句让 OnPipeCreate 函数返回之前，调用 CloseHandle 函数关闭命名管道的句柄，然后将其设置为 NULL，原因前面已经提过了，主要是为了避免程序关闭时，在 CNamedPipeSrvView 对象的析构函数中再次调用 CloseHandle 函数关闭此句柄。

如果成功创建了匿名的人工重置事件对象，那么接下来就定义一个 OVERLAPPED 结构体类型的变量 ovlap，虽然程序中只需要使用该变量的事件对象句柄成员(hEvent)，但是首先应该将 ovlap 变量中所有成员都设置为 0，以免它们影响函数运行的结果，然后将 hEvent 成员设置为刚刚创建的一个有效的人工重置事件对象句柄。

调用 ConnectNamedPipe 函数等待客户端请求到来，该函数的第 1 个参数就是前面调用 CreateNamedPipe 函数返回的一个命名管道句柄，第 2 个参数就是指向 OVERLAPPED 结构体变量的指针，即 ovlap 变量的地址。

如果 ConnectNamedPipe 函数调用失败，则将返回 0 值，但其中有一种特殊情况并不表明等

待连接事件失败了，也就是说，如果这时调用 GetLastError 函数返回 ERROR_IO_PENDING，那么并不表示 ConnectNamedPipe 函数失败了，只是表明这个操作是一个未决的操作，在随后的某个时间这个操作可能能够完成。因此在程序中，当 ConnectNamedPipe 函数返回 0 值时，还应调用 GetLastError 函数，并对其返回值进行判断，如果不是 ERROR_IO_PENDING，则说明 ConnectNamedPipe 函数调用失败，这时提示用户“等待客户端连接失败！”，然后调用 CloseHandle 函数关闭管道句柄和事件对象句柄，并将管道句柄设置为 NULL，之后调用 return 语句返回。

如果上述操作成功了，那么这时调用 WaitForSingleObject 函数等待事件对象(hEvent)变为有信号状态。读者应注意，前面我们已将该事件对象句柄赋给了 ovlap 变量的 hEvent 成员，也就是说，这两个变量 hEvent 和 ovlap.hEvent，现在标识的是同一个对象，因此在调用 WaitForSingleObject 函数时，采用这两个对象中的任一个都是可以的。本例将 WaitForSingleObject 函数的第 2 个参数设置为 INFINITE，即让线程永远等待，直到对象变为有信号状态。

同样的，应该对 WaitForSingleObject 函数的返回值进行判断，如果调用失败，则提示用户“等待事件对象失败！”，然后关闭相关的句柄，并将管道句柄设置为 NULL，之后调用 return 语句返回。

最后，当请求到所有等待的事件对象后，即当该事件对象变成有信号状态时，说明已经有一个客户端连接到命名管道的实例上了。这时，不再需要该事件对象句柄，可以调用 CloseHandle 函数将它关闭。

（2）读取数据

对于命名管道的数据读取操作，与匿名管道的读取操作是一样的，因此可以直接复制已实现的代码，然后将 ReadFile 函数的第 1 个参数修改为本例创建的命名管道的句柄即可，结果如例 8-22 所示。

【例 8-22】

```
void CNamedPipeSrvView::OnPipeRead()
{
    // TODO: Add your command handler code here
    char buf[100];
    DWORD dwRead;
    if(!ReadFile(hPipe,buf,100,&dwRead，NULL))
    {
        MessageBox(“读取数据失败！”);
        return;
}
MessageBox(buf);
}
```

（3）写入数据

对于命名管道的数据写入操作，与匿名管道的写入操作是一样的，所以可以直接复制已实现的代码，然后将 WriteFile 函数的第 1 个参数修改为本例创建的命名管道的句柄即可，结果如

例 8-23 所示。

【例 8-23】

```
void CNamedPipeSrvView::OnPipeWrite()
{
    // TODO: Add your command handler code here
    char buf[100]=”www.baidu.com”;
    DWORD dwWrite;
    if(!WriteFile(hPipe,buf,100,strlen(buf)+1,&dwWrite，NULL))
    {
        MessageBox(“写入数据失败！”);
        return;
}
}
```

至此，完成了利用命名管道实现进程间通信的服务器端程序，利用 Build 命令生成 NamedPipeSrv 程序。

3．客户端程序

这里要创建命名的客户端。同样，可以将客户端程序添加到已有的服务器程序所在的工作区 NamedPipeSrv 中，所以，在此工作区中新建一个单文档类型的 MFC 应用程序，工程取名为 NamedPipeClt，同时设置该工程所在的目录与 NamedPipeSrv 工程的目录平级。

为该工程增加一个子菜单，菜单名称为“命名管道”。再为该子菜单添加 3 个菜单项，并分别为它们添加相应的响应函数，本例选择 CNamedPipeCltView 类接收这些命令响应函数。各菜单项的 ID、名称及响应函数如表 8-13 所示。

表 8-13　添加菜单项及响应函数

ID	菜单名称	响应函数
IDM_PIPE_CONNECT	连接管道	OnPipeConnect
IDM_PIPE_READ	读取管道	OnPipeRead
IDM_PIPE_WRITE	写入数据	OnPipeWrite

为 CNamedPipeCltView 类增加一个句柄变量，用来保存命名管道实例的句柄。

```
private:
    HANDLE hPipe;
```

同样的，在 CNamedPipeCltView 类的构造函数中将其初始化为 NULL。

```
CNamedPipeCltView：：CNamedPipeCltView()
{
// TODO: add construction code here
hPipe=NULL;
}
```

在 CNamedPipeCltView 类的析构函数中，如果判断该句柄有值，则调用 CloseHandle 关闭该句柄。

```
CNamedPipeCltView：：~CNamedPipeCltView()
{
```

```
        if(hPipe)
            CloseHandle(hPipe);
    }
```

（1）连接命名管道

客户端在连接服务器程序创建的命名管道之前，应先判断是否有可以利用的命名管道，这可以通过调用 WaitNamedPipe 函数实现，该函数会一直等待，直到指定的超时间隔已过，或者指定的命名管道的实例可以用来连接为止，也就是说，该管道的服务器进程有了一个未决的 ConnectNamedPipe 操作。WaitNamedPipe 函数的原型声明如下。

```
BOOL WaitNamedPipe(LPCTSTR lpNamedPipeName, DWORD nTimeOut);
```

该函数有两个参数，各自的含义分别如下。

1）lpNamedPipeName：指定命名管道的名称，这个名称必须包含创建该命名管道的服务器进程所在的机器的名称，该名称的格式必须是 “\\.\pipe\pipename”。如果是在同一台机器上编写的命名管道的服务器程序和客户端程序，则当指定这个名称时，在开始的两个反斜杠后可以设置一个圆点，表示服务器进程在本地机器上运行；如果是跨网络通信，则在这个圆点位置处应指定服务器程序所在的主机名。

2）nTimeOut：指定超时间隔。其取值如表 8-14 所示。

表 8-14　nTimeOut 参数取值

取　值	说　明
NMPWAIT_USE_DEFAULT_WAIT	超时间隔是服务器端创建该命名管道时指定的超时值
NMPWAIT_WAIT_FOREVER	一直等待，直到出现了一个可用的命名管道的实例

也就是说，如果这个参数的值是 NMPWAIT_USE_DEFAULT_WAIT，并且在服务端调用 CreateNamedPipe 函数创建命名管道时，设置的超时间隔为 1000ms，那么 WaitNamedPipe 函数将以服务器端指定的 1000ms 为超时间隔。但需要注意的是，对同一个命名管道的所有实例来说，它们必须使用同样的超时间隔。

如果当前命名管道的实例可以使用，那么客户端可以调用 CreateFile 函数打开这个命名管道，与服务器进程进行通信。因此，客户端的连接命名管道的代码如例 8-24 所示。

【例 8-24】

```
void CNamedPipeCltView::OnPipeConnect()
{
  // TODO: Add your command handler code here
  if(!WaitNamedPipe("\\\\.\\pipe\\MyPipe",NMPWAIT_WAIT_FOREVER))
  {
        MessageBox("当前没有可利用的命名管道实例！");
        return;
  }
  hPipe=CreateFile("\\\\.\\pipe\\MyPipe",GENERIC_READ | GENERIC_WRITE,
        0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
  if(INVALID_HANDLE_VALUE==hPipe)
  {
        MessageBox("打开命名管道失败！");
```

```
            hPipe=NULL;
            return;
        }
    }
```

在例 8-24 所示的 OnPipeConnect 函数中，首先调用 WaitNamedPipe 函数，将其超时值设置为 NMPWAIT_WAIT_FOREVER，即让该函数一直等待，直到指定的命名管道的一个实例可以利用为止。对 WaitNamedPipe 函数的返回值进行判断，如果该函数调用失败，即返回值是 0，则提示用户“当前没有可利用的命名管道实例！”，然后立即返回。

如果当前指定命名管道有一个实例可以使用，那么调用 CreateFile 函数打开该命名管道。前面介绍 CreateFile 函数时已经提到，该函数不仅可以对文件进行操作，还可以对管道进行操作。当然，这里指定的文件名就是想要访问的管道名称；为了对该管道进行读取和写入操作，需要同时指定 GENERIC_READ 和 GENERIC_WRITE 两种访问方式；CreateFile 函数的第 3 个参数用来指定共享方式，因为本例中，该管道实例只能接收一个客户端的请求，不需要共享，因此将该参数设置为 0；第 4 个参数用于设置安全性属性，本例将其设置为 NULL；第 5 个参数是指定创建标记，本例将其指定为 OPEN_EXISTING，即打开现有的管道；第 6 个参数是指定文件数，本例将其指定为 FILE_ATTRIBUTE_NORMAL；最后一个参数是指定模板文件，本例将其设置为 NULL。

如果 CreateFile 函数调用失败，返回值是 INVALID_HANDLE_VALUE。因此，如果判断该函数的返回值是 INVALID_HANDLE_VALUE，则提示用户“打开命名管道失败！”，并将管道句柄(hPipe)设置为 NULL，然后立即返回。

（2）读取数据

如果客户端成功打开了指定的命名管道，则可以进行读取和写入操作。这里，可以直接复制上面服务器端已编写的从命名管道读取数据的代码，结果如例 8-25 所示。

【例 8-25】

```
void CNamedPipeCltView::OnPipeRead()
{
  // TODO: Add your command handler code here
  char buf[100];
  DWORD dwRead;
  if(!ReadFile(hPipe,buf,100,&dwRead,NULL))
  {
        MessageBox("读取数据失败！");
        return;
  }
  MessageBox(buf);
}
```

（3）写入数据

这里，可以直接复制服务器端已编写的向命名管道写入数据的代码，但为了加以区分，将客户端写入的数据修改为“命名管道测试程序”，即客户端向命名管道写入数据的代码如例 8-26 所示。

【例 8-26】

```
void CNamedPipeCltView::OnPipeWrite()
{
  // TODO: Add your command handler code here
  char buf[]="命名管道测试程序";
  DWORD dwWrite;
  if(!WriteFile(hPipe,buf,strlen(buf)+1,&dwWrite,NULL))
  {
        MessageBox("写入数据失败！");
        return;
  }
}
```

至此，完成了利用命名管道实现进程间通信的客户端程序，利用 Build 命令生成 NamedPipeClt 程序。

因为采用命名管道实现进程间的通信时，通信的两个进程不需要有任何关系，所以可以独立地运行 NamedPipeSrv 和 NamedPipeClt 两个进程，然后在服务器端选择“命名管道”→“创建管道”选项，创建指定的命名管道，在客户端进程选择“命名管道”→“写入数据”选项连接到这个命名管道；在服务器端选择“命名管道”→“读取数据”选项从命名管道读取数据，这时客户端将弹出一个消息框，提示收到一个网址字符串“www.baidu.com”，程序运行界面如图 8-9 所示。当然，也可以由客户端进程写入数据，服务器进程读取数据，即选择客户端程序的“命名管道”→“写入数据”选项向命名管道中写入数据，然后在服务器端选择“命名管道”→“读取数据”选项从命名管道读取数据，这时服务器端将弹出一个消息框，提示收到“命名管道测试程序”字符串，程序运行界面如图 8-10 所示。

以上就是采用命名管道完成进程间通信的实现，具体过程如下：在服务器端调用 CreateNamePipe 函数创建命名管道之后，调用 ConnectNamedPipe 函数让服务器端进程等待客户端进程连接到该命名管道的实例上。在客户端，首先调用 WaitNamedPipe 函数判断当前是否有可以利用的命名管道实例，如果有，则调用 CreateFile 函数打开命名管道的实例，并建立一个连接。

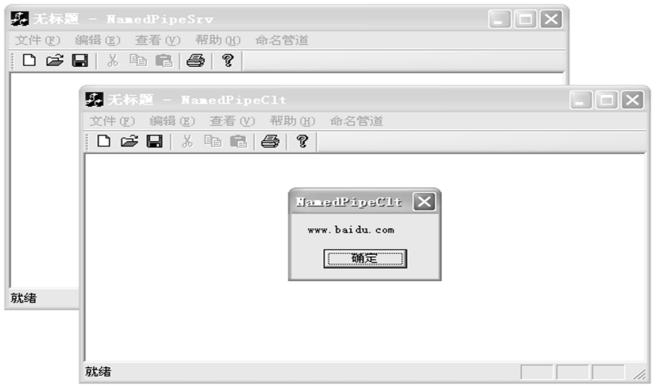


图 8-9　利用命名管道实现进程间通信的程序结果（一）

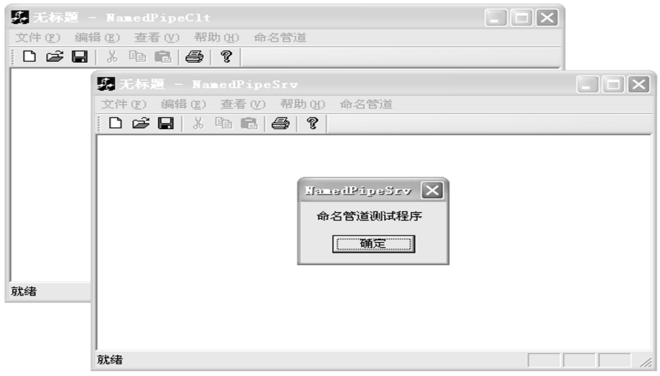


图 8-10　利用命名管道实现进程间通信的程序结果（二）

小结

本章对网络编程的介绍主要集中在线程同步和异步套接字上，读者要了解和掌握网络编程方式的原理，以及常用函数和声明等。还介绍了进程间通信的一些知识，主要包括匿名管道和命名管道，读者需要理解这两类管道的特点及工作方式，也列举了一些常见的实例和常见的编程问题。

第 9 章

通信网编程——OPNET

9.1 仿真技术

随着社会的发展和生活水平的提高，人们对信息的需求越来越多，通信网络需要提供的业务种类越来越多，网络的结构和规模日益庞大，网络的种类也层出不穷，这使得对网络的规划和现有网络的优化至关重要。

面对如此庞大的通信网络，企业在建设网络与开展网上业务之前，最重要的就是要对配置的网络设备、采用的网络技术、承载的网络业务等方面的投资进行综合分析和评估，提出性价比最优的解决方案。如果企业是对现有网络的升级改造，那么企业需要考虑设备升级的兼容性与稳定性，还需要考虑如果采用新的网络技术进行升级，网络的承载能力能否得到改善，用户体验是否达到预期，是否会对其他相关业务带来负面影响；如果网络开通新业务，是否会对网络承载能力带来更大压力。因此，对于企业来说需要搭建接近真实的网络环境与业务，逼真地模拟协议的各个细节，展示网络的测试结果，客观高效地给出网络规划和设计方案是非常最重要的。

长久以来，网络规模和结构比较简单，网络规划与设计主要靠经验来进行。随着网络规模和结构的日益复杂，单纯依靠经验已经无法对新网络进行客观而有效的评估，测试结果不能够真实地反应新协议的表现。这时计算机网络仿真技术作为一种新的网络规划与设计技术脱颖而出，网络仿真不仅可以摆脱传统的靠经验进行网络规划与设计的缺陷，还可以更科学地提供网络规划设计，为新协议的测试结果做出客观、可靠的定量依据，缩短网络建设周期。目前，这种在计算机中构建虚拟的环境来仿真现实的网络环境的技术已经成为网络规划、设计和开发的关键技术。

9.1.1 仿真的定义

网络仿真是一种通过计算机建立网络设备和网络链路的模型，利用数学建模和统计分析的

方法模拟网络承载能力，从而获取网络设计及优化所需要的网络特性参数的一种新技术。

网络仿真的应用包括如下方面。

1）网络仿真根据网络的拓扑结构、路由设计、业务配置等相关参数建立相应模型，模拟网络承载能力，比较不同的设计方案达到的效果。

2）网络仿真为网络建设建立模型，提供性能参数。网络仿真技术的数据获取不同于传统的网络建设，它可以在网络建设之前得到网络的性能参数，对监测数据进行分析处理，为网络建设提供可靠的定量依据。

3）网络仿真可以预测网络容量与服务等级协议。一个合格的网络规划设计师不仅要完成对目前网络的容量设计，还要对未来的网络扩容时间做出准确的预测，这对于将来的网络扩容升级具有重要的意义，同时也是一项艰巨的工作。网络仿真技术可以通过建立实验模型，根据网络用户以及网络增长的趋势进行分析，准确地提供网络性能报告，为后期网络扩容升级降低成本。

4）网络仿真可以进行故障分析。网络仿真系统可以建立各种网络故障模型，分析各种故障造成的影响，提供分析实际网络故障的依据，并提前做好预防措施。

5）网络仿真可以对协议进行分析。数学建模是进行协议分析的重要方法，而仿真实验是验证数学模型的有效手段，因此网络仿真是分析验证协议功能的重要考量工具。

目前，仿真技术应用的领域日益增多，网络仿真软件的应用领域也越来越广泛，开始主要用于网络协议和设备的开发与研究，现在逐渐应用到网络设计和规划。使用群体也由原来的协议研究人员和开发者逐渐增加到网络设计者与规划者。网络仿真软件的开发和应用受到了更多相关技术人员的关注和使用。

9.1.2 仿真的分类

根据不同的划分规则，仿真的分类也有所不同。

1）根据被研究系统的特征分为连续系统仿真和离散事件系统仿真。

2）根据被研究对象的用途分为工程仿真和训练仿真。

3）根据被研究对象的领域划分工程领域仿真和非工程领域仿真。

4）根据被研究对象的种类划分结构仿真、虚拟仿真和实况仿真。

9.1.3 网络仿真

1．网络仿真的背景

随着网络新技术的发展与网络结构的日趋庞大，传统的依靠经验进行网络规划与建设的措施无法有效地反映和预测网络的性能，更不能适应网络设备的研发及新协议的开发。由此，网络仿真技术应运而生。

网络仿真的意义在于提供网络规划与建设前的网络评估，预测网络容量与趋势，降低网络

投资成本，提高网络规划与建设的效率，更加准确有效地进行网络规划设计。

2．网络仿真的应用范围

网络仿真技术因其可以评估现有的网络性能、验证其中的配置错误，制定优化及升级前的网络规划方案，并且可以对下一代网络进行仿真设计等功能，被广泛应用于各种通信系统的设计、规划及运营维护。除了可以用在商用和军用的通信网络中，还可以对单一网络和混合网络进行技术分析。

3．网络仿真关心的问题

网络仿真是一项建立于数学模型和统计分析方法之上的工作，为了保证证仿真结果的正确性，模拟仿真网络系统工作应注意以下步骤及问题。

1）理解仿真系统、明确仿真目的。在进行仿真之前，要清楚仿真系统的结构以及构成系统的模块之间的关系，明确仿真目的及要研究的问题。

2）明确仿真模块对象，抓住主要问题。网络仿真模型的建立要能反映主要问题所在，化繁为简、突出重点。

3）确定网络模型、定义输入和输出参数。网络模型可行性分析的关键在于网络仿真中是否能够准确地确定业务量或数据量的概率分布、流量大小。这就要求建模者能够熟练地使用网络建模工具，熟悉系统参数、变量及其之间的相互关系，并根据系统仿真结果进行分析。网络仿真模型需要明确其输入和输出参数，输入和输出参数指网络业务或数据量。

4）确定输入。在确定的网络模型下输入网络业务数据。

5）可信度与模型的完善。模型的可信度取决于建立的模型能否真实地反映出即将仿真的网络、所用信息的先验知识和实验数据是否正确完备、将数学模型转化为仿真算法的仿真模型是否完善等。因此，对模型进行可信度的检验与改善是不可或缺的。

6）检验仿真结果是否足够详细解释所研究的问题。网络仿真是一个变仿真过程为数值实验的过程，其本质是采用离散事件的概率对事件进行分析。根据网络仿真的结果分析要求解决的问题，如果仿真结果不能支撑要解决的问题，则需要调整建立的模型和输入的数据量等。

7）结果是否有效。为了从运行阶段所产生的数据中找出系统运行规律，对仿真系统的性能做出评价，为系统方案的最终决策提供辅助支持，建模者需要对结果是否达到统计的稳定状态有一个明确的结论。仿真模型运行后所产生的数据需采用统计学的分析方法，对仿真数据的可靠性、一致性、置信度等做出判定，并将仿真结果以动画、曲线、图表和文字形式形成仿真报告或论文。

4．常用的仿真流程

建立准确的模型是一个循环过程，需要仿真者在理解系统和仿真目的基础上建立模型，再对仿真结果进行分析，并且在结果出现问题时对模型进行审核并做出准确的判断与修改。图 9-1 所示为一般的仿真流程。

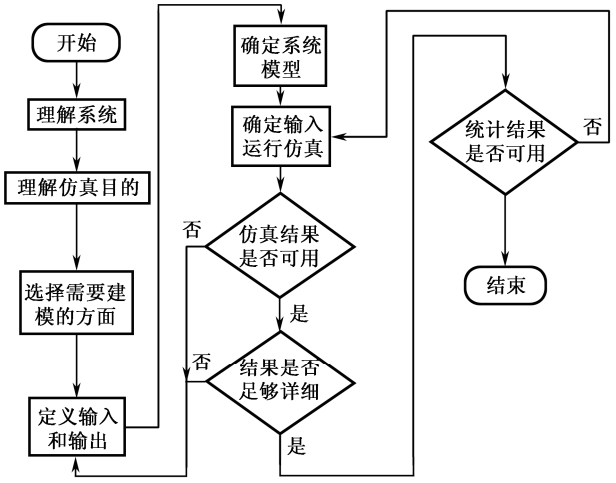


图 9-1　仿真流程图

9.1.4　仿真工具

1．MATLAB

其主要功能如下。

1）数值分析与数字信号处理。

2）通信系统设计与仿真。

3）数字图像处理。

4）数值和符号计算。

5）工程与科学制图。

6）控制系统的设计与仿真。

7）Simulink 动态仿真。

Simulink 是 MATLAB 最重要的组件之一，它提供一个动态系统建模、仿真和综合分析的集成环境。它不需编写程序，只需要通过简单直观的鼠标操作，即可构造出复杂的系统，广泛应用于控制理论和数字信号处理的复杂仿真和设计。

2．NS

NS（Network Simulator）是一个由 UcBerkeley 开发，可以对无线网、有线网、卫星网、局域网和广域网等不同的网络形式进行仿真的软件。它支持 DSR、AODV、DSDV 等无线网络协议；支持 FTP、Telnet、CBR and VBR 等高层业务；还支持 TCP、UDP 协议。

3．SPW

SPW（Signal Processing Worksystem）是美国 Cadence 公司提供的能对数字信号处理及通信系统算法进行开发、仿真、调试并进行性能估计的信号处理工作系统。它利用 System View 组

件，可以构造各种复杂的模拟、数字、数模混合系统及各种多速率等线性、非线性系统，用户只需从 System View 配置的图标库中调出有关图标并进行参数设置，完成图标间的连线，然后运行仿真操作，即可以时域波形、眼图、功率谱等形式给出系统的仿真分析结果。

SystemView 的基本功能如下。

1）搭建多速率系统和并行系统。

2）仿真现代通信系统及所有 DSP 领域。

3）可进行滤波器与线性设计。

4）进行可视化动态仿真。

5）提供子系统模块功能。

6）能够与 MATLAB、CCS 等多种软件扩展使用。

7）可进行信号分析和数据块处理。

9.2　OPNET 概述

OPNET 是当前网络仿真及分析领域中较优秀的软件，为通信网络仿真和优化，以及网络高效地管理提供了整套解决方案。本节将介绍 OPNET 产品。

9.2.1　OPNET 历史和现状

OPNET 公司成立于 1986 年，起源于麻省理工学院。1987 年发布了第一款具有预测网络性能和管理的优化工具。它在 2002 年被列为福布斯全美最佳 200 中小型企业，目前已在 Nasdaq 上市。

OPNET 公司由最初的 OPNET Modeler 一种产品，发展到拥有 Modeler、ITGuru、SPGuru、WDMGuru、ODK 等一系列产品。

OPNET 公司目前在全球客户超过 2500 个，其中大约 75%的客户在美国本土，大约 25%的客户在其他国家。其客户群主要如下。

1）通信设备制造商：Cisco、Nortel Networks、3COM、Lucent 等。OPNET 公司为这些客户提供网络设备、协议及其应用的开发工具。

2）大中型企业：BOEING、Daimler Benz 等具有庞大内部网络传输的企业。

3）电信运营商：AT&T、NTT DoCoMo、France Telecom 等具有复杂的网络结构和协议配置的运营商。

4）OPNET 由于其仿真的精确性、友好的界面及高级体系结构，被军方和政府的研发机构所采用。

5）OPNET 还向欧美等西方国家的大专院校提供免费软件，支持和资助大专院校的科研教学。

目前，OPNET 公司以其出众的技术赢得了许多第三方测评机构的奖项，成为业界领先的智能化网络仿真、分析、管理解决方案的提供商。

9.2.2 OPNET 的系列产品

OPNET 公司的产品包括 Modeler、ITGuru、SPGuru、OPNET Development Kit 及 WDMGuru。产品不同，面向的客户也不同。

1）Modeler 的作用是加速网络研发，主要面向研发客户。

2）ITGuru 适用于智能化的网络设计、规划和管理，主要面向大中型企业。

3）SPGuru 内嵌了流分析模块、网络医生模块、多提供商导入模块、MPLS 功能模块等，成为电信运营商智能化网络管理、规划及优化的平台。

4）WDMGuru 提供了管理 WDM 光纤网络功能，并为测试产品提供了一个虚拟的光网络环境，适用于光纤网络的运营商和设备制造商。

5）OPNET 开发包 ODK 为开发环境、NetBizODK 为运行环境，是一个更底层的开发平台，可以用于设计用户自定义的解决方案，定制用户的界面，并且 ODK 提供了大量的函数，用于网络优化和规划。

（1）Modeler

OPNET Modeler 为开发人员提供了业界领先的网络技术开发环境，因其可以用来加速研发过程，开发大型网络而被应用于设计和通信网络、设备、协议和应用。

使用 Modeler 的益处如下。

1）提升网络研发成果。研发人员可以使用 Modeler 提供的专门的编辑器、分析工具和特定的模型进行开发，避免了在其他不必要的模块上浪费时间。

2）改善产品质量。Modeler 可以在开发之前提供建模，得到产品参数，为产品设计提供参考数据。

3）缩短开发周期，使产品在实际研发之前得到验证。

Modeler 可以支持所有的网络类型，实际网络和网络组件的结构也可以通过其面向对象的建模方法和图形化的编辑器直接映射到模型中。

Medeler 的主要特性如下。

1）层次化的网络模型。网络拓扑结构由多个子网嵌套而成。

2）简洁的建模方法。Modeler 建模过程分为 3 个层次，模拟单个对象的行为在过程层次中完成，设备互连在节点层次中完成，设备互连组成网络在网络层次中完成。这种明确的分工机制，更加有利于项目的分工管理。

3）有限状态机。用户可以使用 C/C++语言对过程层次的协议和其他过程进行模拟，控制仿真的详细程度。Modeler 编程的核心是由 C/C++语言以及 OPNET 本身提供的 400 多个库函数，统称为 Proto C 语言。

4）全面支持协议编程。Proto C 内部嵌入了多种协议，无须开发者进行代码编写。

5）开源系统。Modeler 中的代码是开源的，用户可以在标准协议上添加或者修改已有代码来进行自己的实验。

6）高效的仿真引擎。由于 Modeler 可使用 C/C++语言进行代码编写，因此使用 Modeler 仿真平台，效率较高。

7）集成分析工具。Modeler 仿真平台可以将结果导出到第三方软件中，也可以仿真线性与非线性的曲线。

8）动态。Modeler 仿真平台可以演示仿真过程中的动态画面。

9）集成的调试器。OPNET 集成了自己的调试工具——OPNET Debugger（ODB），能够快速地发现仿真中的漏洞。在 Windows 平台下能够联合 VC 进行调试。

（2）ITGuru

ITGuru 代表了 OPNET 公司产品在网络规划、管理及优化方面的地位，为现在的网络人员提供了良好的技术管理平台。

ITGuru 可以智能识别网络的路由器、交换机、协议、服务器及其所支持的业务，可以提高企业管理员发现和解决问题的能力。

使用 ITGuru 的益处如下。

1）节约成本。ITGuru 对 IT 的预算能够使投资者在理解性能的前提下，投入较少的资金获得最大的回报。

2）提高运营效率。ITGuru 为技术工程师提供结构化的、可用的诊断和验证功能，提高工程师发现和解决技术问题的能力。

3）提高企业生产率。ITGuru 可以指出应用的瓶颈，支持平滑过渡来优化应用的响应时间，从而提高企业的生产率。

4）减小风险。ITGuru 可以通过仿真故障和过载情况来减小网络故障出现的风险。

ITGuru 技术通过构建拓扑结构、流量和业务的配置来进行文本诊断、网络规划和优化，保障了数据的精确性，能够真实地反映现实网络。网络管理员、规划人员和操作人员可以通过 ITGuru 构建的虚拟网络进行有效的故障诊断、变更验证，为网络升级做出准确规划。

OPNET 的 ITGuru 可以将管理网络、服务器及网络运行整合起来，使网络获得较好的端到端性能和较高的可用性。企业 IT 部门可以使用 ITGuru 作为预测应用软件、设备及网络配置在做出修改之后对网络性能产生的影响，从而发现决定网络性能的根本因素。**错误!未找到引用源。**为企业 IT 部门的工作范畴。

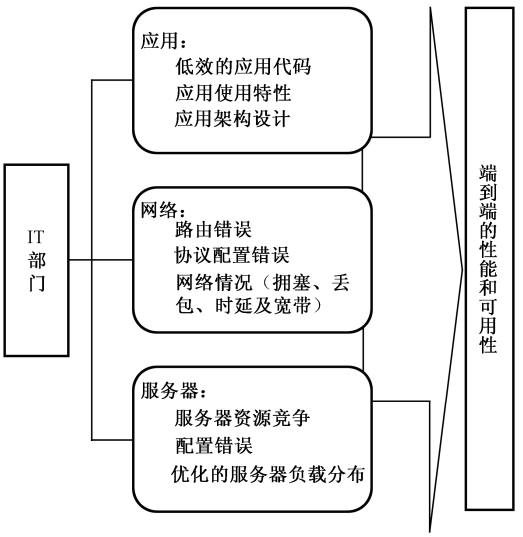


图 9-2　企业 IT 部门的工作范畴

（3）SPGuru

SPGuru 是通过对交换机、路由器、协议和数据流的建模，帮助客户进行网络规划、排错、验证及部署流量工程的面向网络服务提供商的智能化网络管理软件。

使用 SPGuru 的益处如下。

1）赢利。SPGuru 可以使运营商精确地预测网络的变化和增长所带来的影响，在加速提供和使用新业务的同时保持原来的业务质量，从而在保持现有客户的同时为用户提高高质量的业务。

2）减少运营成本。SPGuru 提供了强大的、自动的网络设计功能，使运营商能够更好地管理带宽和设备，从而获得较大的回报。

3）提高运营效率。SPGuru 为技术工程师提供结构化的、可用的诊断和验证功能，提高工程师发现和解决技术问题的能力。

4）减小风险。SPGuru 可以通过模拟各种场景，预测各方面的风险，从而设计出更加稳定、成熟的网络。

整体来说，SPGuru 与 ITGuru 相比功能更加齐全，工作流程相似。表 9-1 所示为 SPGuru 的独有功能。

（4）WDMGuru

WDMGuru 的主要功能如下。

1）利用 WDMGuru 可以进行波分复用光纤网络的分析和测评，从而设计出健全且节约成本的光纤网络。

2）WDMGuru 提供详细的硬件成本信息和配置信息，展示规划网络不同层面的变更。

3）WDMGuru 可以帮助运营商计算发生故障的可能性，并预测相应的损失来减少投资风险。

表 9-1　SPGuru 的独有功能

模块	SPGuru 独有功能
Flow Analysis	IS-IS
MPLS	不限制 BGP 个数 自动的 MPLS 流量控制
NetDoctor	IS-IS 配置验证规则 MPLS 配置验证规则（计划中）
Multi-Vendor Import(MVI)	支持 Lucent NavisCore 路由器导入 支持 MPLS 路由器配置导入 支持 IS-IS
包含的模块	ESP、MPLS、PNNI、IP Multicast、IPv6*、Circuit Switched、Advanced Server

4）WDMGuru 提供虚拟网络环境，可进行 SDH 层及光层网络的最优设计。

5）WDMGuru 可以规划业务的增长、优化当前网络的配置及设计新的网络。

（5）ODK 以及 NetBiz

ODK（OPNET Development Kit，OPNET 开发包）提供大量的 API 和优化函数、对话编辑器、图表编辑器及 ETS 文件，可以帮助用户设计软件与建立界面。用户使用 ODK 设计的软件相比 Modeler、ITGuru、SPGuru、WDMGuru，更具有针对性，能够帮助用户建立属于自己的网络仿真、分析、优化及管理的软件。

ODK 开发包包括 ODK 库和 ODK 工具两大类。ODK 库提供用户界面、图形图像、导入/导出及优化算法等方面的十几类 API。ODK 提供用于设计的优化算法函数的软件。除此之外，ODK 编辑器还提供项目编辑器、节点编辑器、过程编辑器、对话框编辑器等。图 9-3 所示为 ODK 的组成。

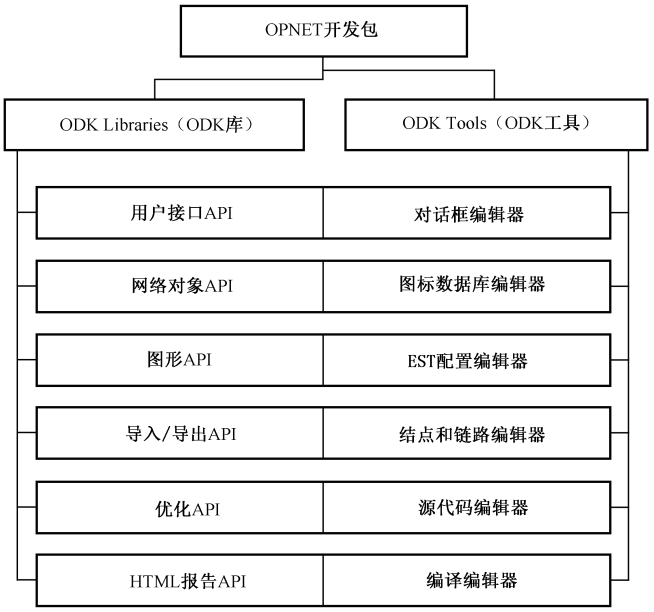


图 9-3　ODK 的组成

9.2.3　OPNET Modeler 仿真平台

在 OPNET 各种产品中，Modeler 针对不同领域，表现出不同用途。

1）对于企业网的模拟，Modeler 通过组建标准模型，从业务、网络、服务器 3 方面捕捉重要的流量进行分析，从而完成对网上交易、数据库等业务时间慢于正常时间的情况进行评估。

2）对于运营商网的模拟，Modeler 使运营商通过对整个业务层、流量的模拟，查出业务配置中的错误。

3）针对研发的需求，Modeler 为用户提供了建立新协议和配置、模拟细节的开放环境。本书侧重于将深层次的细节精确模拟体现出来。

准确地将模拟的特点体现出来，这是传统方式不能做到的。

Modeler 应用包括端到端结构、系统级的仿真、新的协议开发和优化、网络和业务层配合。端到端结构应用技术可以从 IPv4 网络升级为 IPv6；新协议的开发，如目前的 4G 网络协议；系统级仿真能够分析新路由或调度算法如何使路由器、交换机达到 QoS；通过 Modeler 模拟网络和业务配置之间的矛盾，找出解决方案，从而达到网络和业务层之间配合的最好性能。

Modeler 具有收集分析统计量、查看动画和调试等功能，收集各个网络层次的性能统计参数，编制和输出仿真报告，如图 9-4 所示。

Modeler 仿真分为 6 个步骤：网络拓扑（Topology）、业务配置（Traffic）、结果统计（Statistics）、运行仿真（Simulation）、重复调试模块仿真（Re-simulation）、发布结果和拓扑报告（Report）。

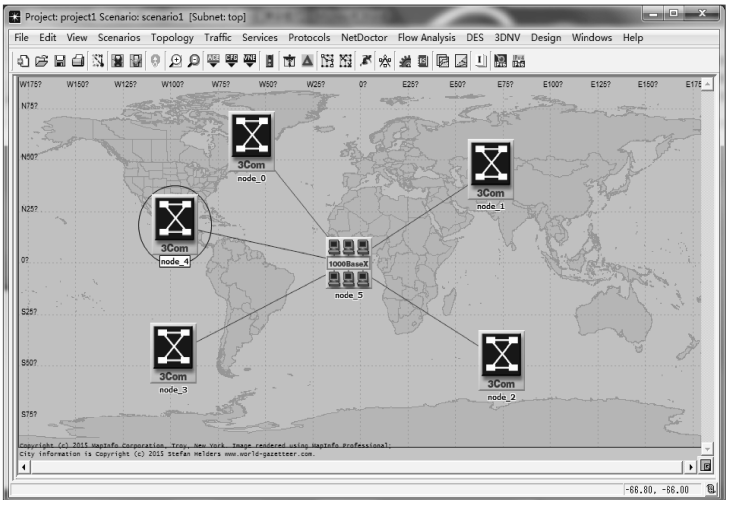


图 9-4　Modeler 层次化的建模机制和集成的建模、仿真及分析环境

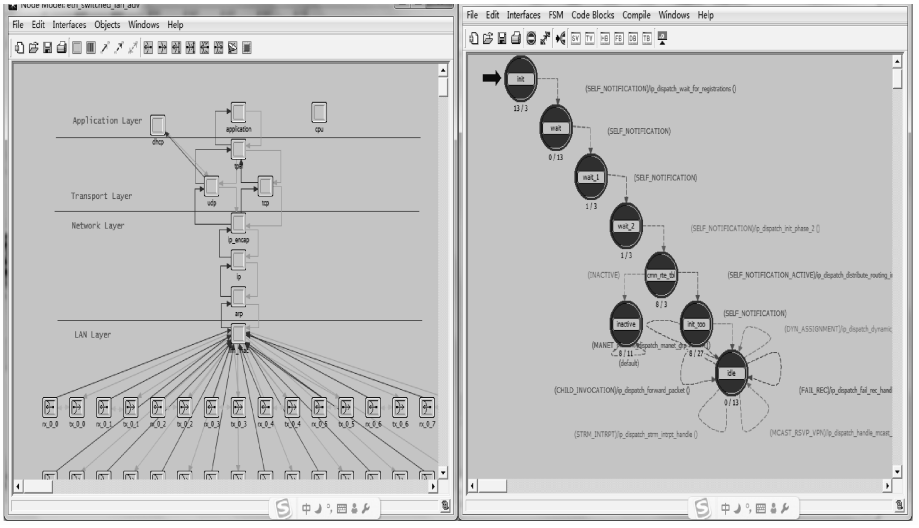


图 9-4　Modeler 层次化的建模机制和集成的建模、仿真及分析环境（续）

9.3　OPNET Modeler 运行环境

OPNET Modeler 使用了面向对象的建模方式和图形化的编辑器，不仅能反映出实际网络，还能反映出网络组件的结构，它提供的开发环境能支持通信系统和分布式系统。正因为 OPNET Modeler 灵活的层次建模方法，它才能支持所有与网络研究相关的通信设备与协议。

9.3.1　系统需求

OPNET Modeler 所需的硬件及软件环境如表 9-2 所示。

表 9-2　OPNET Modeler 硬件及软件环境

CPU 系列	内存（最低/建议）	硬盘需求（最低/建议）	操作系统
Sun Microsystems SUN-4 CPU：SPARC 系列（如 UltraSPARC）	最低 128MB 建议 256MB 或更高	最低 350MB 建议 550MB 或更高	Sun Solaris 2.6/7/8/9
PC 兼容 CPU：Intel 奔腾系列，500MHz 或更高			Windows NT/2000/XP
Hewlett-Parkard 9000/7xx & 9000/8xx CPU：PA7000 V1.1c 或更高			HP-UX11.0

Modeler 不能正常启动的常见原因是操作系统没有安装补丁或安装不正确，所以安装 OPNET Modeler 之前要先给操作系统正确安装补丁，如表 9-3 所示。

表 9-3　OPNET Modeler 所需补丁

厂商	操作系统	补丁名称
Sun Microsystem	Solaris 2.6	105591-09　或更新
	Solaris 7	106327-06　或更新
	Solaris 7-64bit	106300-07　或更新
	Solaris 8	108434-07　109147-16
Microsoft	Windows NT	Service Pack 3、5（不支持 4 和 6）
	Windows 2000	支持 Service Pack 1 和 2，但不是必需的
HP	HP-UX11.0	PHSS 21906

9.3.2　OPNET Modeler 安装

因为 OPNET Modeler 本身不含有编辑器，所以在安装 OPNET 之前需要在 Windows 平台上安装 Microsoft Visual C++ 6.xx 或 Visual Studio.NET。

安装好后要注意检查环境变量的设置，环境变量的设置在“控制面板”→“系统”→“高级”→“环境变量”中进行，如图 9-5 所示。

其中，VC 注册的环境变量如表 9-4 所示。

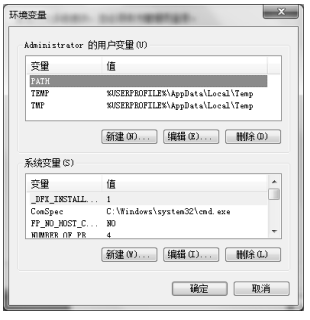


图 9-5　“环境变量”对话框

表 9-4　VC 注册的环境变量

变量名	Include
变量值	C:\Program Files\Microsoft Visual Studio\VC98\atl\include C:\Program Files\Microsoft Visual Studio\VC98\mfc\include C:\Program Files\Microsoft Visual Studio\VC98\include
变量值	C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib C:\Program Files\Microsoft Visual Studio\VC98\lib
变量名	MSDevDir
变量值	C:\Program Files\Microsoft Visual Studio\Common\MSDev98
变量名	Path
变量值	C:\Program Files\Microsoft Visual Studio\Common\Tools\WinNT C:\Program Files\Microsoft Visual Studio\Common\MSDev98\bin C:\Program Files\Microsoft Visual Studio\Common\Tools C:\Program Files\Microsoft Visual Studio\VC98\bin

安装好即可安装 Modeler。Modeler 的安装盘一共有 3 个部分，第 1 部分是应用程序，第 2 部分是模型库，第 3 部分是在线文档，依次安装即可。

9.3.3　OPNET Modeler 开发环境

Modeler 是用网络层、节点层、进程层 3 层建模机制来描述系统模型的，所以 Modeler 也提供了相应的编辑器（网络编辑器、节点编辑器、进程编辑器）来刻画这 3 个层次的模型。为了方便整个建模的过程，Modeler 还提供了如包编辑器、图标编辑器等其他编辑器。文件类型如图 9-6～图 9-8 所示。

其中：

1）Project、Node Model、Process Model：分别为项目编辑器、节点编辑器和进程编辑器，分别对应 OPNET 的 3 层建模域。

2）Link Model：有线链路模型编辑器，用于设置链路的传输速率、链路上运行的包格式及管道阶段的模块。

3）Path Model：用来显示虚电路、虚路径的路径编辑器。

4）Demand model：背景流模型，用来设置网络背景流与应用背景流。

5）ETS Source：外部系统支持工具，可以在 OPNET 面板上定义新的功能键。

6）External Source：定义外部文件。当两个或两个以上进程模块用到了某个函数时，应该将其定义成外部函数包含到外部文件中。

7）Header file：显示进程模型中的头文件，并允许对其进行操作。

8）Pipeline Stage：管道阶段模型，用来描述物理层，OPNET 的 3 种链路都由相应管道阶段组成。

9）Network Model：网络模型编辑器。因为在 OPNET 先前版本中没有 Project 编辑器，所

以会使用该编辑器编辑网络。

10）Probe Model：定义用来收集统计量的探针模型。一般在项目上右击，在弹出的快捷菜单中选择需要的统计量，在 Node Statistics 中有很多统计量可以选择。如图 9-9 所示，首先选择 ATM 组别，它们就是在节点模型中定义的，现在看到的名称是后来提升的。

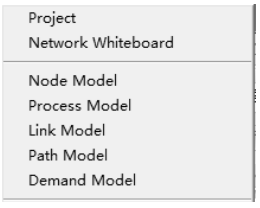


图 9-5　文件类型列表（1）

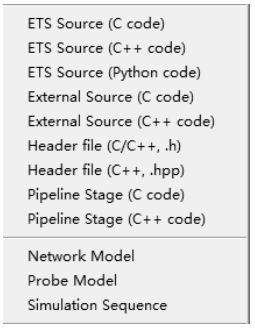


图 9-6　文件类型列表（2）

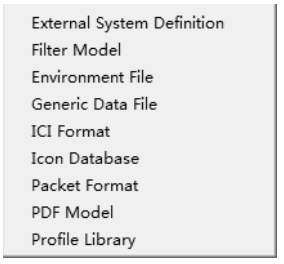


图 9-8　文件类型列表（3）

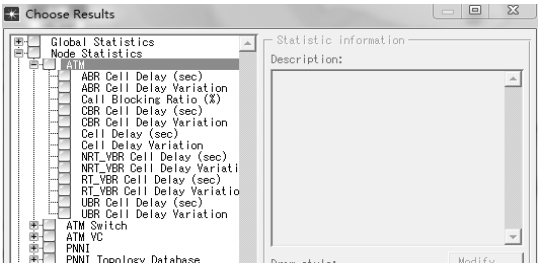


图 9-9　选择统计量组别与提升后的统计量

Probe Model 可以收集到各种未提升到网络层的统计，具体地讲就是单击相应的统计功能键来收集节点、链路、路径、背景流量、配对物件、属性的统计。一般统计量以时间为自变量，而属性探针可以使属性变量成为自变量。另外，自定义动画也需要在探针模型中设置。

11）Simulation Sequence：高级仿真配置文件，用来定义仿真参数，该文件与高级仿真属性对话框的设置是相关联的。它可以创建一个包含多个仿真子集的仿真序列，而每个仿真子集也包含很多仿真。当有仿真在进行时，可以单击 Stop Sequence 停止所有方针，单击 Stop Set 停止仿真组，单击 Stop Run 来停止当前仿真。

12）Extern System Definition：定义 OPNET 与外部系统交互的信息格式，如外部系统传输数据的类型与从哪个系统成员开始传输都需要 Extern System Definition 来定义，它以模块的形式放在节点模型中，像一个回字形图标。

13）Filter Model：自定义结果过滤器。OPNET 已经提供了很多过滤器对已经得到的仿真结果进行加工处理，使结果更加突出重点或更容易被人理解。也可以组合基本的过滤器或定义全新的过滤器并对其进行编译，这样就创建了新的过滤模块。

14）Environment File：定义外部环境文件。它主要有两个作用：一个作用是如果下一个仿真和上一个仿真的运行环境完全一样，则可以直接复制环境文件；另一个作用是如果只有一个

Modeler license，则在运行当中不能编辑模型，这时可以执行环境变量文件来进行仿真，即op_runsim -net_name<网络模型名称>-duration<仿真持续时间>-ef <环境变量文件名>，它只占用仿真执行 license，而不占用 Modeler 的 license，这样不仅可以提高仿真速度，还可以保存为静态仿真文件*.sim。

15）Generic Data File：描述网络的文本文件。读取这种特殊的文本文件需要 OPNET 中标准的核心函数，这些函数也可以检验内存和做一些预处理工作。

16）ICI Format：接口控制信息格式编辑器。ICI 可以在节点模型中从一个进程模块中把一些控制信息传输到另一个进程模块中。但是其与包格式信息的不同在于，包格式传输信息必须使用封包流，而 ICI 可以绑定任何类型的中断。除此之外，为了和实际情况更加接近，很多标准协议也采用 ICI 交换握手信息。

17）Icon Database：图标库文件，在文件库中包含多种图标以表示网络设备，该文件库支持自定义图标。

18）Packet Format：封包格式编辑器，可以对封包域的类型、颜色等进行设定，可以设定为 Information 类型和 Packet 类型，前者使该域只作为承载封包信息的标签不计入包长，后者则可实现包的封装，该域的大小则于封包的大小。

19）PDF Model：概率分布编辑器。OPNET 通常要使用很多分布概率，除去 OPNET 本身提供的标准概率分布函数外，该编辑器还可以自定义概率分布函数：一种离散概率分布，一种连续概率分布。离散概率分布如图 9-10 所示，假设封包的目的地有 4 种选择，图中发往第 1 个节点的有 50%的封包，发往第 2 个节点的有 17%的封包，发往第 3 个节点的有 23%的封包，发往第 4 个节点的有 10%的封包。连续概率分布需要设置得非常精确，否则可能导致 EMA 文件再次编辑。

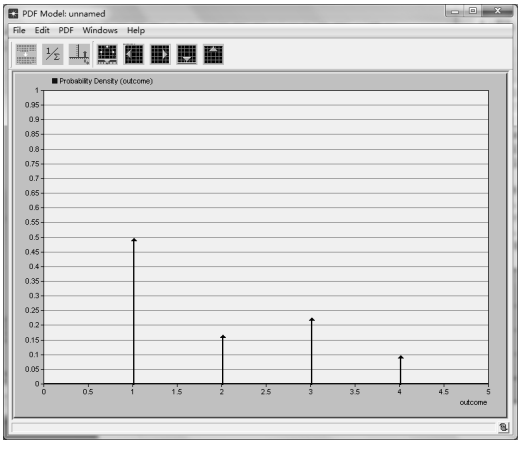


图 9-10　PDF Editor 编辑的离散概率分布

20）Profile Library：对应高级仿真属性中的“Profiling”选项卡。

9.4　OPNET Modeler 编程基础

9.4.1　Modeler 编程概述

（1）状态变量的定义

状态变量可以在进程唤起和函数调用的过程中保持原值。通过查看 Modeler 提供的定义状态界面的源文件，可以发现它与标准的 C/C++定义变量是一样的。如图 9-11 所示，单击状态变

量对话框中的“Edit ASCII”按钮即可看到定义状态变量的源代码，而出现的数据类型就是 OPNET 特有的数据类型。

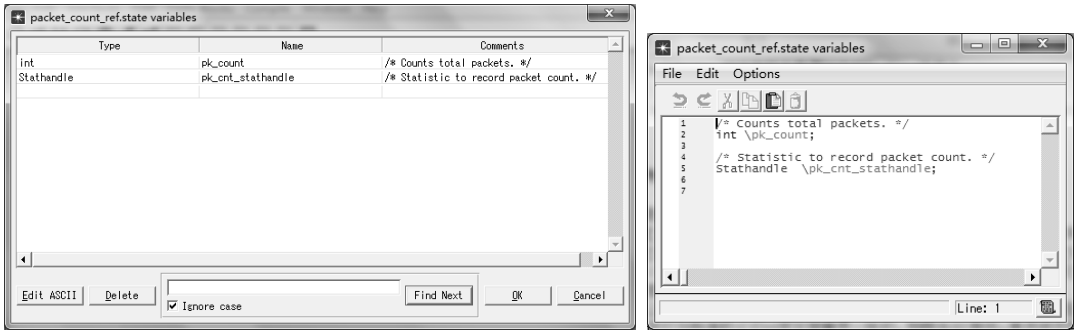


图 9-11　状态变量的定义和源代码

（2）临时变量定义

临时变量和状态变量不同之处在于，临时变量在进程唤起的过程中不再保持原值。临时变量和传统 C/C++中的变量在进程模型中定义时是一样的，如图 9-12 所示。

（3）头区及头文件的使用

进程模型中的头区用于定义数据结构、宏、常量和声明函数等。如图 9-13 所示，头区声明了包含的头文件，定义了宏、常量并声明了函数。

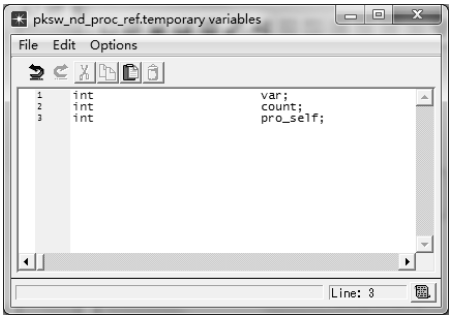


图 9-12　临时变量定义

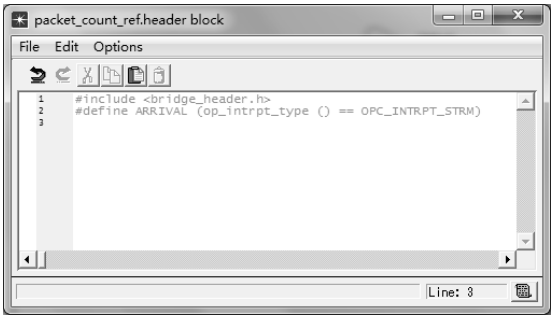


图 9-13　头区的定义

头文件与头区的定义差不多，区别在于头区定义的函数是变量且只适用于本进程模型，而头文件定义的大多是一个系统或多个进程所公用的变量、数据结构和函数等，使用它们时只需要在头区中包含该头文件即可。

（4）函数区及外部文件

头区和头文件声明了函数，但相应函数的具体定义还需要在函数区或外部文件进行。图 9-14 显示了函数在函数区中的定义。函数在函数区中定义时，函数的入口与出口用 FIN 和 FOUT 表示，这样有利于编译和定位错误位置。

进程模型中的外部文件用外部代码编辑器进行编写和编译，文件名为 xxx.ex.c，编译后的文件名为 xxx.ex.o。

外部文件编译后需要在进程模型中进行声明。如图 9-15 所示，选择“File”→“Declare External Files…”选项，然后选择需要的外部文件即可。

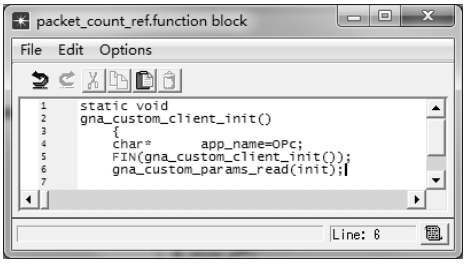


图 9-14　函数区中的函数定义

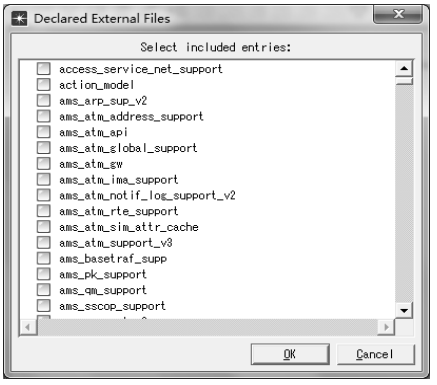


图 9-15　外部文件

9.4.2　Modeler 文件操作

模型文件包含了建立、保存、打开、更新和删除等文件操作。和 Windows 操作系统的文件操作相似，但 OPNET Modeler 的文件操作也有特殊性。

1．文件的建立和保存

OPNET Modeler 的系统界面 File 提供了不同格式文件编辑器的入口，可以根据需要建立相应的模型文件。

用户所建立的模型文件可以保存在自定义的默认文件夹中，如果希望将文件保存到新文件夹中，则需要进入“Add Model Directory…”界面，单击新建文件夹后添加目录，在 OPNET Modeler 中才能看见该文件夹。

2．文件的打开

对于任意编辑器，在没有特别指定目录的情况下打开的目录文件都是默认目录文件。没有注册过的目录需要通过 File→Model File→Add to Model Directory→选中要注册的目录文件，如果所选中的目录包含子目录，则要选中“Include all subdirectories”复选框完成源路径的添加，如图 9-16 所示。

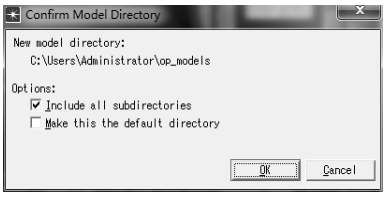
3．文件的删除

OPNET Modeler 文件的删除分为项目删除和其他类型文件的删除。

（1）项目删除

删除项目文件通过“File”→“Model File”→“Delete Projects”选项进行。

（a）浏览文件夹



（b）选中复选框

图 9-16　添加注册目录

（2）其他文件类型的删除

删除文件需要先选择文件类型，然后选择具体文件，通过“File”→“Model File”→“Delete Models”选项完成。图 9-17 所示为删除文件对话框。

如图 9-18 所示，要删除输出的标量文件“cct_a_ref”，在弹出的删除文件对话框中找到输出标量文件“Output Scalars（.os）”，选中展开后要删除的目录，单击“Delete”按钮即可删除文件。

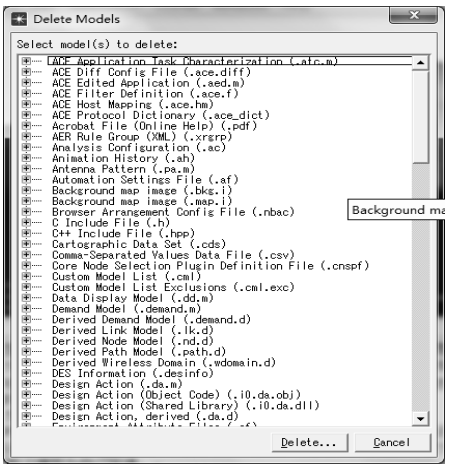


图 9-17　删除文件对话框

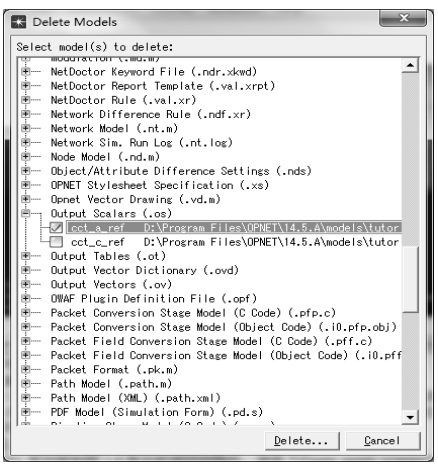


图 9-18　标量文件的删除

OPNET Modeler 是一个基于层次化和模块化设计的仿真软件，删除文件可能会导致不同领域的变化。例如，删除项目文件不会影响节点模型，而删除节点域的节点模型会影响使用该节点模型的项目文件。同样，改变进程模型也会影响对应的节点域或网络域的模型。

4．文件的命名

由于 OPNET Modeler 自带很多模型，因此用户在命名文件时应使用自己容易识别的符号。一般命名方法是“<initial>_文件名”。其中，“<initial>”表示文件的版本、文件特征等。本书以 book 或 b 作为标记，以标注模型为自己所建。

9.4.3 实例：建立简单星型网络

首先明确进行网络仿真的目的。通过网络仿真研究网络扩容以后的网络性能的变化。

1. 建立模型背景

（1）建立一个新场景

1）建立新项目。选择“File”→“New”→“Project”选项，单击“OK”按钮，弹出如图 9-19 所示对话框。在其中输入项目名称和场景名称，如命名项目名称为“book_star”，场景名称为“campus”，然后单击“OK”按钮即可。

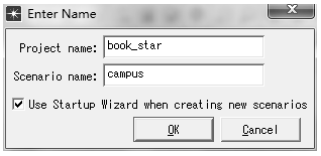


图 9-19　建立项目名称及场景名称

2）在弹出的如图 9-20 所示的项目向导内，选择“Create empty scenaio”选项，单击“Next”按钮，选择网络范围。

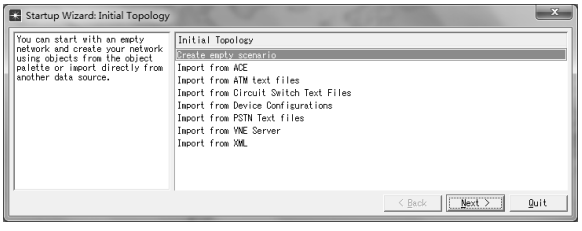


图 9-20　建立空场景

3）确定建模背景和范围，如图 9-21 和图 9-22 所示，选择以“Campus”为背景，在 300×300Meters 范围内建立局域网，单击“Next”按钮。

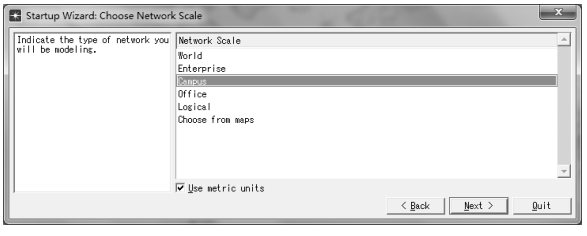


图 9-21　确定建模背景

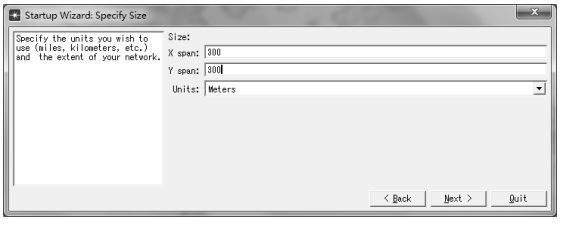


图 9-22　确定建模范围

（2）选择建模对象

1）如图 9-23 所示，将 Model Family 中的 Sm_Int_Model_List 模块对应的 Include 改为“Yes”。

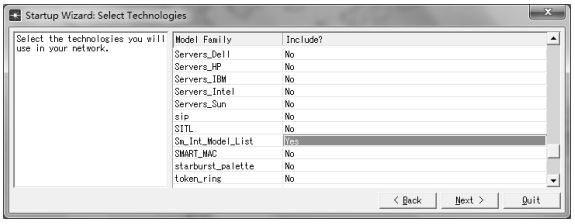


图 9-23　选择模块对象

2）如图 9-24 所示，再次确定环境设置。单击“Finish”按钮后，将打开如图 9-25 所示的建模场景。建模场景就是建立模型时所用的背景，可以是一定范围的地图，也可以是不包含任何对象的空白范围。

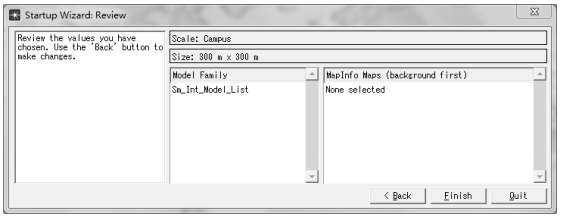


图 9-24　建模环境确认

2．建立网络拓扑

（1）选择建立的网络拓扑模式

网络拓扑结构可以单个节点逐次添加，也可以通过配置参数快速添加。下面用快速配置来构建网络拓扑。

1）如图 9-26 所示，选择“Topolog”→“Rapid Configuration”选项。

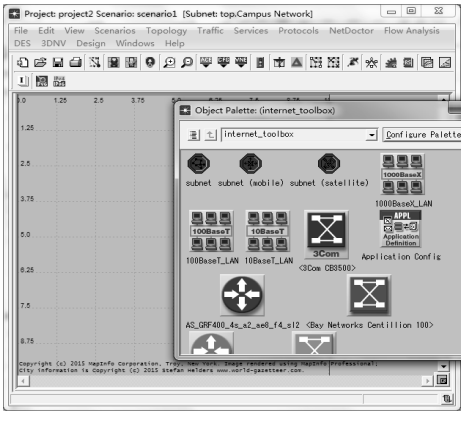


图 9-25　建模场景

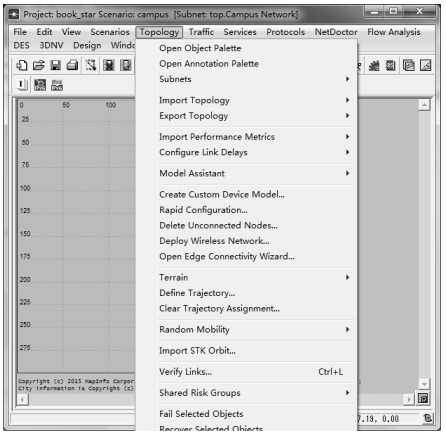


图 9-26　快速配置网络拓扑

2）如图 9-27 所示，在快速注册下拉列表中选择“Star”选项，然后单击“Next”按钮。

（2）设置网络节点参数

如图 9-27 所示，单击“OK”按钮后将弹出如图 9-28 所示的设置 Star 网络节点对话框。

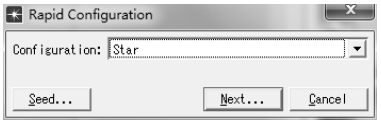


图 9-27　选择网络拓扑类型

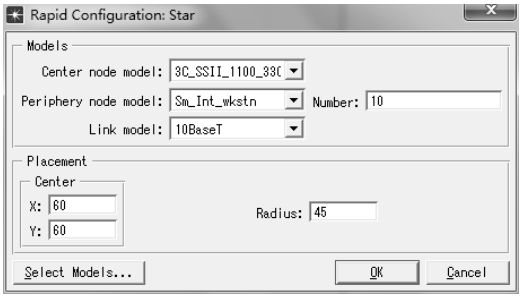


图 9-28　配置网络拓扑参数

其中：

1）Center node model 表示星型网络中心节点的设备类型。

2）Periphery node model 表示星型网络的端节点设备类型。

3）Link mode 表示从中心节点到端节点连接介质的类型。

4）Number 表示网络中端节点的数目。

5）Placement 表示星型网络中心在场景中的位置。

6）Radius 表示星型网络中心到端节点的长度。

按图 9-28 所示参数设置好后，单击“OK”按钮，在网络空间中将出现如图 9-29 所示的简单网络拓扑模型。

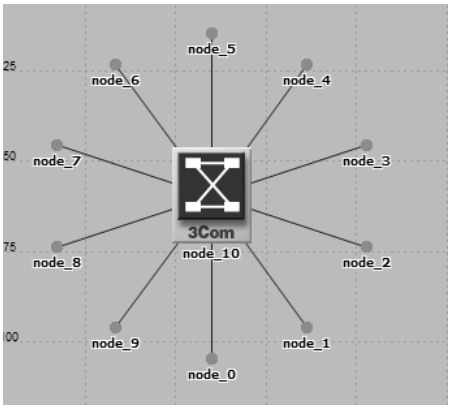


图 9-29　简单网络拓扑模型

（3）引入服务器

在对象面板中搜索“Sm_int_server”并添加到项目空间中，并搜索出 10Base-T 链路对象进行服务器与交换机之间的连线。

3．添加业务流

在 OPNET 中构建好的网络模型也需要运行如 Web Server、FTP Server 等各种业务。用户可以通过多种配置方式添加网络业务，本实例采用标准的端到端业务配置方式完成。

1）增加业务的配置：

在对象面板中搜索 Application Definition 模块和 Profile Definition 模块并添加到工作空间中，右击“Edit Attribute”，将其更改为“Application Config”和“Profile Config”。由于 SM_int_server 对象平台中已经完成了配置工作，所以不用进行手动配置。

2）单击服务器，此时服务器周围出现圆圈，如图 9-30 所示，表示对象被选中。右击，在

弹出的快捷菜单中选择“Edit Attribute”选项，弹出对话框，更改命名为“server”。

在 File 中选择“Save”选项，保存项目，此时完成如图 9-31 所示的网络拓扑，然后关闭对象模板。

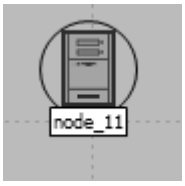


图 9-30　选中对象

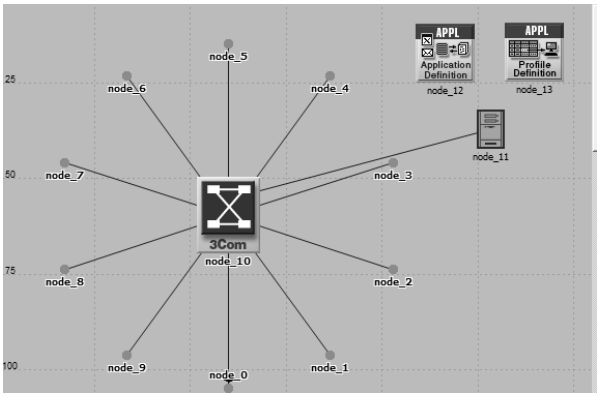


图 9-31　配置完成后的网络模型

4．运行仿真及结果观察

在运行仿真之前先要选择需要测量的量。本实例要得到两个统计量，分别是服务器负荷与整个网络扩容前后的时延。

（1）选择仿真统计量

1）服务器仿真统计量：选中服务器后右击，在弹出的快捷菜单中选择“Choose Individual DES Statistics”选项，弹出如图 9-32 所示的对话框：展开“Ethernet”→“Load”，然后单击“OK”按钮，完成服务器统计时延的定义。

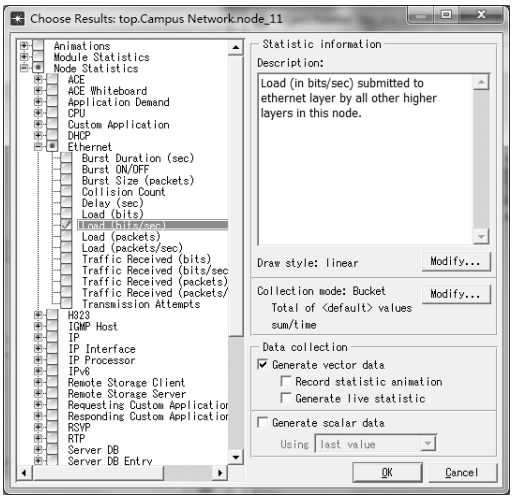


图 9-32　选择服务器统计量

2）全局统计量：在网络模型的工作空间中右击（不要右击对象），在弹出的快捷菜单中选择“Choose Individual DES Statistics”选项；弹出如图 9-33 所示的对话框，展开“Global Statistics”→“Ethernet”→“Delay”，然后单击“OK“按钮。

（2）选择仿真时间

运行仿真有两种途径：一是在菜单中选择“DES”→“Configure/Run Simulation”选项，二是在工具栏中单击运行仿真按钮。操作后会弹出如图 9-34 所示的仿真运行对话框。在“Duration”文本框中设置“0.5”，即虚拟网络运行 0.5 小时。

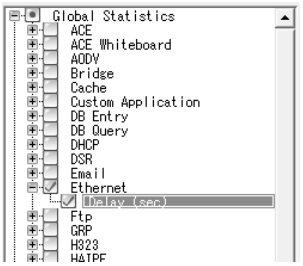


图 9-33　选择全局统计量

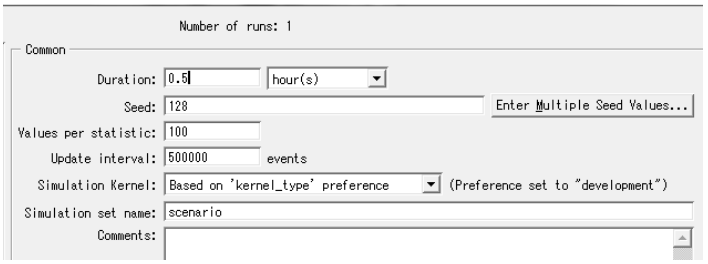


图 9-34　仿真参数的设置

单击“Run”按钮运行仿真，运行结束后关闭对话框。

（3）显示结果

选择“DES”→“Results”→“View Statistics”选项，将打开仿真结果的树形图，如图 9-35 所示。从树形图中选择需要观察的对象即可。

单击“View Result”对话框中的“Show”按钮，将结果单独显示。图 9-36 和图 9-37 所示分别为服务器的负载和时延情况。

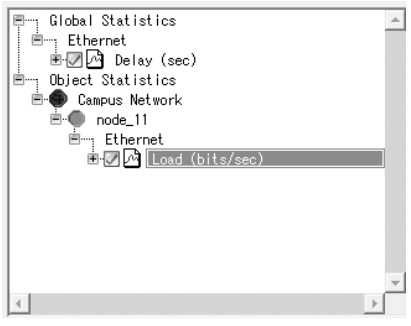


图 9-35　观察仿真结果

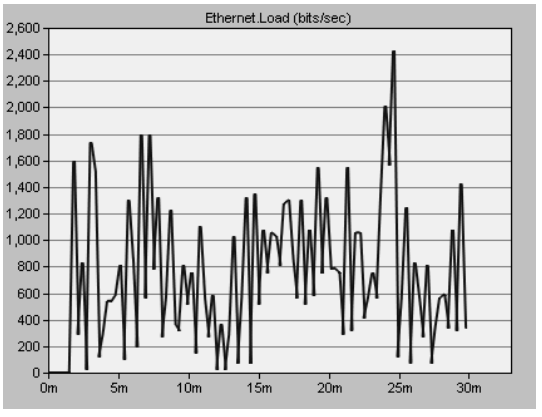


图 9-36　服务器负载统计结果

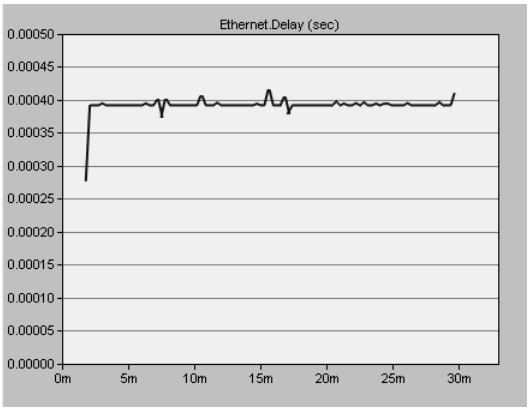


图 9-37　服务器时延统计结果

5．扩建网络

网络扩建是在同一个场景下进行的，所以比较扩建前和扩建后网络性能的变化，应复制场景保持相同场景下前后的仿真结果。

（1）复制场景

选择“Scenarios”→“Duplicate Scenario”选项，并命名新场景为“campus1”。

（2）建立扩建网络拓扑

按住鼠标，并拖动图 9-29 所示的部分，此时将出现如图 9-38 所示的情况。

选择“Edit”→“Cope”→“Paste”选项，将复制的网络放在 X 轴为 230，Y 轴为 150 处，右击，在弹出的快捷菜单中完成局域网的复制和粘贴。打开对象面板，拖动 Cisco2514 交换机进入工作空间，并将用 10Base T 的线构成如图 9-39 所示的网络拓扑。选择“File”→“Save”选项，保存结果。

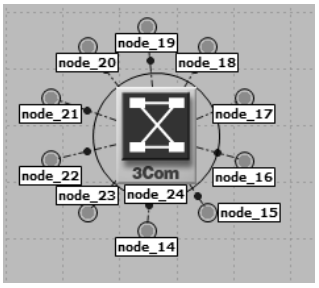


图 9-38　选中复制对象

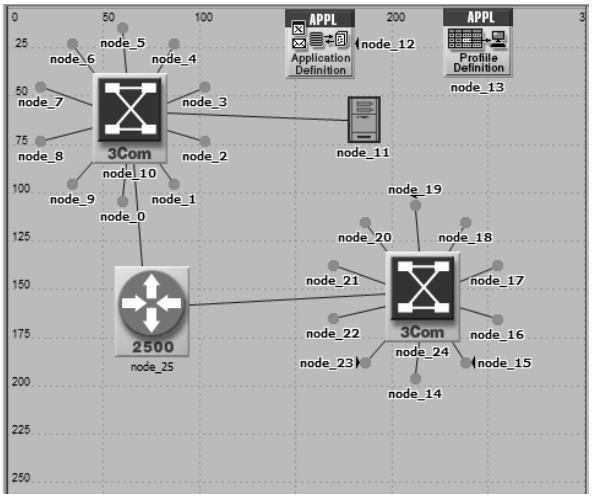


图 9-39　扩展网络拓扑

6．运行仿真

选择“DES”→“Configure/Run Simulation”选项，或者在工具栏中单击运行仿真按钮，仿真时间（Duration）仍然选择 0.5。

7．比较结果

（1）输出未处理的结果

1）选中服务器节点并右击，在弹出的快捷菜单中选择“Compare Result”选项。

2）展开“Campus Network.server”→“Ethernet”→“Load”，在比较对话框中选择“All Scenarios”选项。

3）单击“Show”按钮，查看比较结果，如图 9-40 所示。从结果上看，扩展网络后服务器的最大负载达到 4700b/s，较未扩展前增大一倍。

4）选择“DES”→“Result”→“Compare statistiics”选项，在弹出的对话框的树形图中展开“Global Individual Statistics”→“Ethernet”→“Delay”，如图 9-41 所示，两个网络平均时延的仿真结果没有太大的区别。

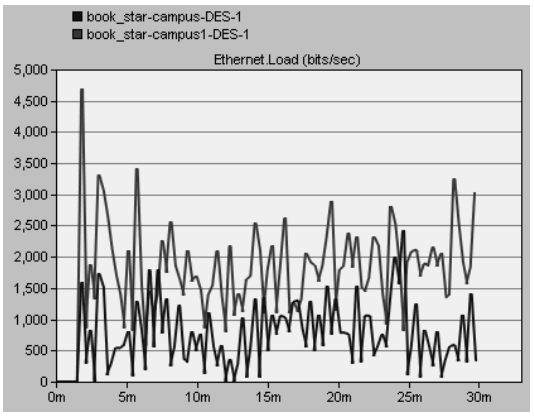


图 9-40　服务器负载结果比较

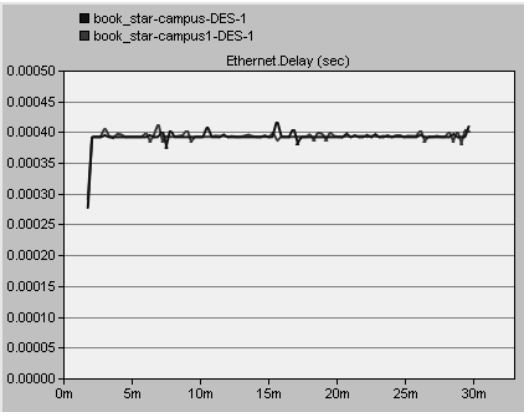


图 9-41　端到端时延仿真结果比较

（2）比较时间平均统计值

为了可以更明显地比较出扩容前后网络的性能变化，可以对仿真结果进行处理，在下拉列表中选择“time_average”选项，并单击“Show”按钮，可以出现基于时间的平均统计结果，如图 9-42 所示。

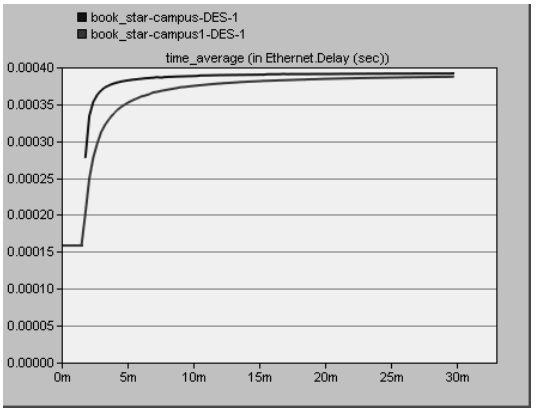


图 9-42　选择时间统计值

8．分析结果

从图 9-40～图 9-42 中可以看出，增加一个局域网后，网络时延并没有太大的改变，平均访问时延为 0.0003s；而服务器负载却增加很多，平均为 1200b/s～2500b/s。这为网络管理者提供了一个参考，网络硬件上的改进主要是更新服务器。

9．观察网络的层次结构

1）选中服务器并双击，显示如图 9-43 所示的服务器和硬件及软件的建模结构。从该节点域模型可以看出，它与 TCP/IP 协议构架类似。

2）单击节点域中的 TCP 模块，显示如图 9-44 所示的 TCP 进程域模型。进程域模型是一个有限状态机，单击一个状态模块，如 init，能够得到如图 9-45 所示的进程执行代码。

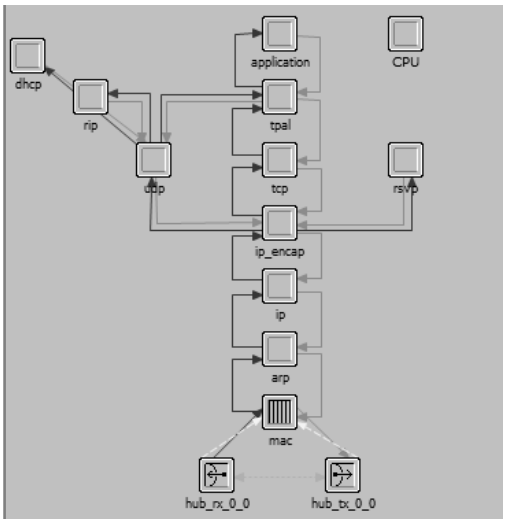


图 9-43　服务器的节点域模型

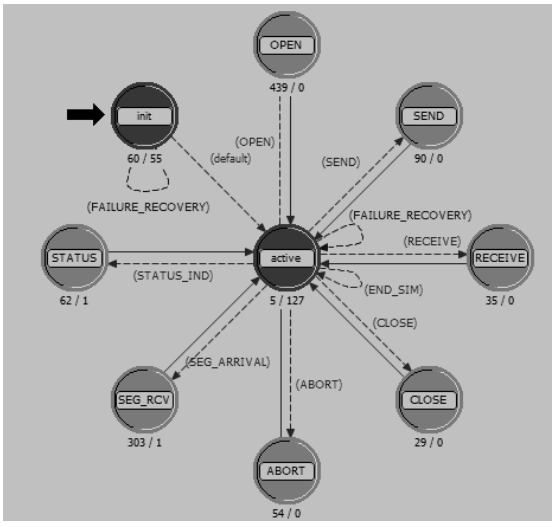


图 9-44　TCP 模块的进程域模型

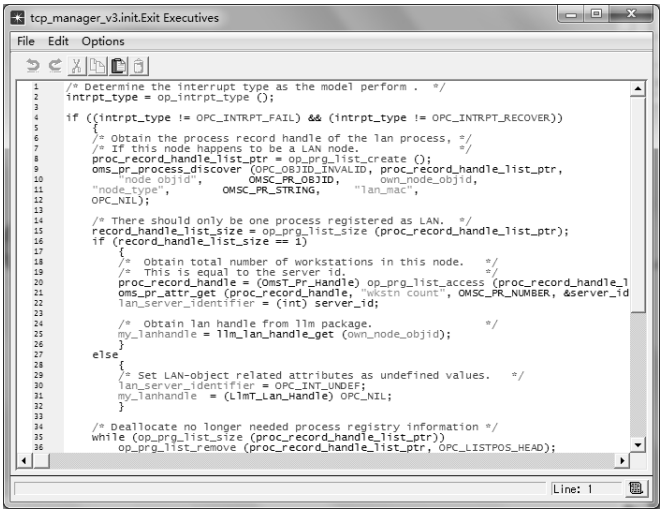


图 9-45　执行代码

9.5　OPNET 程序调试

OPNET Modeler 具有很强大的调试功能，其中 ODB 是 OPNET 的仿真调试器，可以为用户提供灵活控制仿真的方式。ODB 的目的是分析程序执行的环境。用户可以交互地控制仿真，获取时间和对象的信息。

ODB 的命令作用如下。

1）执行单个或者多个事件。

2）对特定的事件、时间、模块和进程设置断点。

3）跟踪核心函数的执行情况。

4）显示当前仿真实体的信息。

5）显示当前内存的使用。

6）通过修改对象属性来影响仿真的执行。

下面对 ODB 的基本概念进行简单介绍。

（1）断点

断点是对应 ODB 的控制过程，如前所述，ODB 执行期间有“开”和“关”两个状态。“开”即仿真正常运行；“关”即对应仿真断点的概念，意味着仿真产生终止，被称为“断点”。

（2）跟踪

ODB 的跟踪显示了核心程序的名称、变量类型、变量值、返回值和对事件调用的反应过程，也被用于观测进程逻辑在动态仿真中的执行情况。设置在模块上的跟踪也包括模块中所有进程的信息和其子进程的调用及返回。

（3）映射

ODB 的映射是把以某些标准收集来的仿真事件建立成的列表。列表反映出每个选择实体的 ID、拓扑结构等信息。映射可以为实体 ID 提供 OBD 命令。动态实体的映射可以更加准确地观察仿真状态。

（4）诊断区

诊断区包含在进程模块中，由 C 语言构成并通过 ODB 命令激活。诊断区模块命令与用户自定义命令的不同之处在于调用方式的不同，诊断区可以直接被 ODB 命令调用，而用户必须在中断传递到包含用户自定义时才能完成调用。

（5）标签

标签是一种方法，用来标注特定事件并进行跟踪。打开标签后可以打印状态和临时变量，也可以打印字符串等信息。

9.5.1 查看 OPNET 日志文件

在 OPNET 仿真结束后，可以从产生的两种日志文件中得到一些提示和错误信息，这两种日志文件分别是仿真日志（Discrete Event Simulation log）和错误日志（Error log）。

在工程空间上右击，在弹出的快捷菜单中选择“Open DES Log”选项可以打开仿真日志文件，如图 9-46 所示，其中的内容是调用 OPNET 函数 op_prg_log_handle_create() 和 op_prg_log_entry_write()写入的。

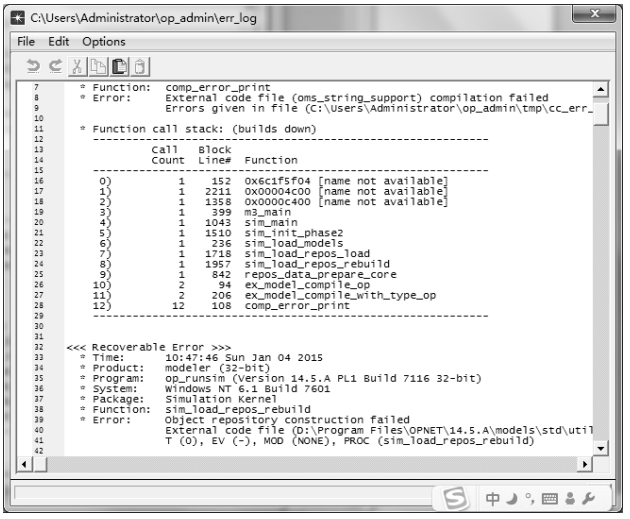

C:\Users\Administrator\op_admin\err_log
File Edit Options
* Function: comp_error_print
* Error: External code file (oms_string_support) compilation failed
Errors given in file (C:\Users\Administrator\op_admin\tmp\cc_err_
* Function call stack: (builds down)
Call Block
Count Line# Function
0) 1 152 0x6c1f5f04 [name not available]
1) 1 2211 0x00004c00 [name not available]
2) 1 1358 0x0000c400 [name not available]
3) 1 399 m3_main
4) 1 1043 sim_main
5) 1 1510 sim_init_phase2
6) 1 236 sim_load_models
7) 1 1718 sim_load_repos_load
8) 1 1957 sim_load_repos_rebuild
9) 1 842 repos_data_prepare_core
10) 2 94 ex_model_compile_op
11) 2 206 ex_model_compile_with_type_op
12) 12 108 comp_error_print
<<< Recoverable Error >>>
* Time: 10:47:46 Sun Jan 04 2015
* Product: modeler (32-bit)
* Program: op_runsim (Version 14.5.A PL1 Build 7116 32-bit)
* System: Windows NT 6.1 Build 7601
* Package: Simulation Kernel
* Function: sim_load_repos_rebuild
* Error: Object repository construction failed
External code file (D:\Program Files\OPNET\14.5.A\models\std\util
T (0), EV (-), MOD (NONE), PROC (sim_load_repos_rebuild)

图 9-46　仿真日志文件

OPNET 标准协议模块不仅仅会在出现错误时发出提醒，也会将可能发生的各种行为记录下来。

打开错误日志文件有两种方式：一是在“Help”菜单中打开错误日志文件（“Help”→“Error Log”→“Open”），错误日志文件以文本方式保存为<home>/op_admin/err_log；二是在 OPNET 控制台（console）窗口输入“op_vuerr”命令查看。错误日志文件包含了函数调用堆栈信息，可以从函数阶层性的调用关系中精确定位出错位置。

```
        ------------------------------------------------------
        <<< Program Abort >>>                                                    --错误类型
          * Time:          13:16:01 Wed Jan 07 2014
          * Product:     modeler (32-bit)
          * Program:     op_runsim (Version 14.5.A PL1 Build 7116 32-bit)
          * System:       Windows NT 6.1 Build 7601
          * Package:     Simulation Kernel
          * Function:    sim_load_repos_load
          * Error:        The set of models necessary for running the simulation is incomplete.     --出错原因
                            Check that all the 'repositories' attributes are complete.
            T (0), EV (-), MOD (NONE), PROC (sim_load_repos_load)        --出错时间、事件、进程模块、出错函数

          * Function call stack: (builds down)
            ------------------------------------------------------------
                        Call     Block
                        Count    Line#   Function
            ------------------------------------------------------------
             0)             1      152   0x14444c0d [name not available]
             1)             1     2211   0x00004c00 [name not available]          --可忽略的内存消息(0-2)
             2)             1     1358   0x0000c400 [name not available]
```

```
        3)          1      399   m3_main
        4)          1     1043   sim_main
        5)          1     1510   sim_init_phase2                    --仿真核心程序调用堆栈(3-7)
        6)          1      292   sim_load_models
        7)          1     1830   sim_load_repos_load
       ------------------------------------------------------------
```

以上是一条完整的错误日志信息，其中包括错误类型、出错原因、出错时间、错误事件、出错所在进程模块、函数调用堆栈及最终出错函数。为了获取准确的函数调用堆栈信息，必须使用 FIN、FOUT、FRET 等界定函数范围的标识符来编写函数。在 FIN 后漏掉 FOUT 或 FRET 就会出现如下错误：

```
<<<Program Abort>>> Standard function stack imbalance
```

如果程序庞大复杂，则要在使用外部文件中层层调用子程序。如果子程序没有配对 FIN 和 FOUT，则可能出现日志信息无法给出正确的出错位置等问题，只会给出由于函数堆栈不平衡不断积累最终导致内存泄漏时的程序的位置，因此编写程序时切记使 FIN 和 FOUT/FRET 配对。

9.5.2 使用 OPNET Debugger 调试

要想产生 ODB 调试信息，就必须把仿真核心类型设定为 development。仿真的核心为了加快速度，必须优化从而不产生 ODB 调试信息。此外，仿真属性中包含 debug 环境变量。

ODB 为控制和管理仿真行为提供了一个交互式环境。ODB 能跟踪并显示诊断信息，还可以支持断点定义。要想实现 ODB 功能，必须有相应的程序支持，ODB 中有很多指令可以做激活调试状态的依据，这些指令可以在 ODB 窗口中输入“help”指令进行查看，如图 9-47 所示。

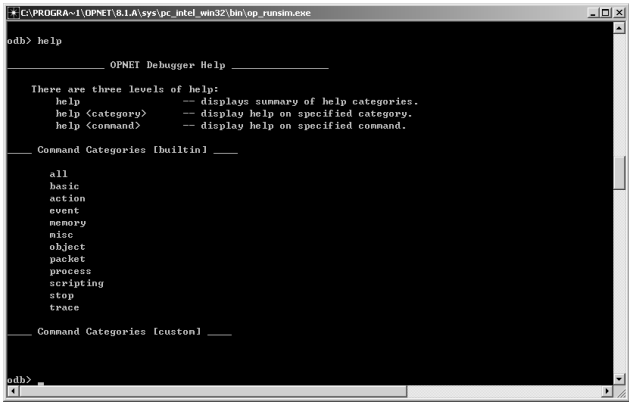


图 9-47 “help”指令带来的参数列表

ODB 常用的指令可分为 basic、event、object、packet 等几类。basic 类指令主要是一些基本的操作；event 类指令主要是对事件进行操作；object 类指令主要是对节点、信道等对象进行操作；packet 类指令处理所有与包相关的操作。

ODB 调试的主要内容是针对进程的调试，即对定位进程、控制进程、跟踪进程和显示进程

进行调试。

（1）定位进程

1）promap <Objid>显示指定进程模块包含的所有的 Process ID。

2）promap all 显示所有仿真中存在的 Process ID，如图 9-48 所示。

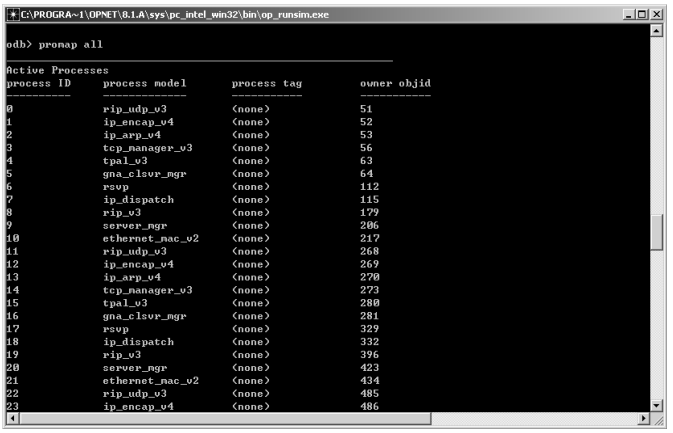


图 9-48　输入“promap all”指令的结果

（2）控制进程

在 ODB 调试过程中输入“prostop <proc_id>”指令，无论什么进程，仿真都会暂时中断。通过 op_prg_odb_bkpt(label)可以在进程模型中指定位置设置断点并用标签标识，在调试过程中如果激活了这个标签，则仿真会在这个位置中断。

（3）跟踪进程

1）protrace <proc_id>跟踪并显示调用指定进程的信息。

2）ltrace <label>是最实用的调试指令，如果想找到程序中的逻辑错误，则可以在可疑的程序段中加入这样一个语句：

```
if (op_prg_odb_ltrace_active("label")==OPC_TRUE){ printf(...)}
```

打印需要观察的变量，在 ODB 中激活标签即可看到仿真中这些变化的情况。

3）proltrace <proc_id> <label>显示指定进程中设置过某个标签的位置所对应的信息。

（4）跟踪进程

1）在进程的诊断块中编写与调试相关的程序后，ODB 调试时能通过 prodiag <proc_id>执行诊断块对应的程序，从而显示调试信息。

2）proldiag <proc_id> <label>指令相当于 prodiag <proc_id>与 ltrace <label>的叠加。

9.5.3　OPNET 与 Visual C++联合调试

ODB 调试与 Visual C++(VC++)调试相比更侧重于逻辑上的调试，ODB 调试只显示函数的原型和参数的结果，在 fulltrace 下也只能显示函数的调用情况和代码的返回值，而 VC++调试可以显示一些有关赋值、比较的代码。所以才把 ODB 调试用于全局的错误定位，而 VC++调试用

于局部精细的跟踪程序，查看变量的变化，或进入函数查看细节。

VC++提供了一个功能非常强大的调试环境，不仅可以支持设置断点、观察变量，还可以处理单步跟踪、跟踪子程序等操作。OPNET 与 VC++联合调试可以分为：设定环境变量，设定 OPNET 参数，首次联调选择 OPNET 强制编译，绑定 OPNET 仿真进程，观察变量这 5 个步骤。

前面已经介绍了如何设置环境变量，现在介绍如何设定 OPNET 和 VC++联合调试的参数。

在项目编辑器中选择“Edit”→“Preferences”选项，如图 9-49 所示，修改以下属性值。

1）在 bind_shobj_flags（动态连接）和 bind_static_flags（静态连接）的值后面加上/DEBUG，此作用是连接时将所有的目标（*.obj）文件集成为一个动态链接库（*.dll）文件，同时加入调试信息。

2）在 comp_flags 和 comp_flags_cpp 后面加上/Zi /Od，此作用是在编译时产生调试信息，并且在调试时关闭编译器的优化功能。

OPNET 与 VC 联合调试的步骤如下。

首先要设置仿真属性，选择“Simulation”→“Configure Simulation (Advanced)”选项，设置后选择“Envirement files”，将其中 debug 的属性值变为“included”，运行处于编译状态的仿真会打开 ODB 窗口；也可以根据情况的发展强制编译所有的进程模型，只要将 force_compile 的属性值改变为“included”即可。具体操作如图 9-50 所示。

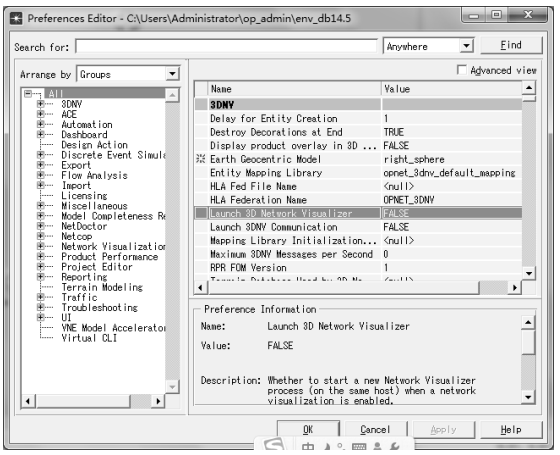


图 9-49　与调试相关属性的设置

设置 VC++以便于调试 OPNET 进程，方法是打开 VC++，选择“Debug”→“Attach to Process”选项，然后选择“op_runsim_dev.exe（或者“op_runsim_opt.exe”）”选项，这样可以用 VC 来完成 OPNET 程序的调试。当 Attach to Process 选项框为空时，这不是软件本身的问题，可能是一些其他软件的进程和 op_runsim_dev.exe 进程冲突，如杀毒软件，这时可以打开任务管理器，对 op_runsim_dev.exe 进程进行调试，这样即可启动 VC++并自动连接 OPNET。

用 VC++观察变量需要引用指针 op_sv_ptr 来查看而不能直接观察到，指针指向所有的状态变量。例如，在查看变量窗口中输入（*op_sv_ptr）A 就是要观察状态变量 A。

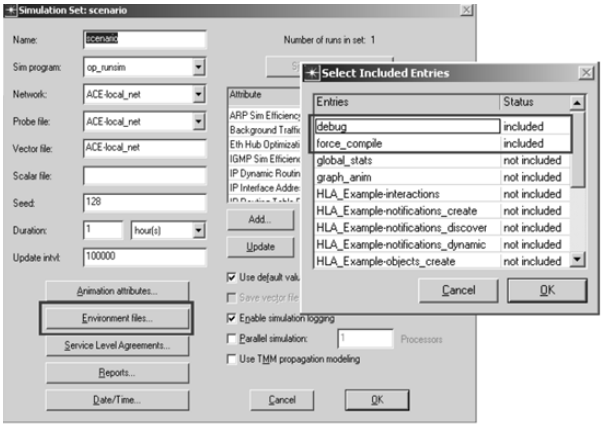


图 9-50　与调试相关的仿真属性

9.5.4　ODB 常见错误及问题

在仿真报告中，有以下几种类型的错误。

1）Program Abort Problem：程序异常中断，在仿真过程中禁止出现。

2）Reccoverable Error Problem：可恢复错误，仿真过程中不禁止出现，但会导致取消当前运行。

3）Diagnostic Error Problem：诊断错误，是可恢复的较小错误，在 diag_enable 设置为 TRUE 时能产生报告，默认值为 FALSE。

4）Warning Message：警告信息。

常见错误如下。

（1）内存无效访问错误

```
<<< Program Fault >>>
program abort – Invalid Memory Access
```

出现上述错误一般是程序中指针出现了问题，解决办法如下。

1）按照上一节所示，在“Edit”→“Preference”中给 comp_flags 加上/Od /Zi 字段，给 bind_shobj_flags 加上/DEBUG 字段。

2）在“Edit”→“Preference”中找到 handle_exception，将 TRUE 改为 FALSE，这样即可用 VC++来调试程序中的错误。

在运行仿真时如果发生了异常，则可以单击 Cancel 按钮，自动打开 VC，一般 debug 会停留在发生异常的地方，但是有时候也会停留在其他地方，这时可以通过查看发生异常的事件的 event_id 值来寻找错误所在的地方，在 ODB 调试的时候使用 evstop 指令可以在程序出错前被终止，然后用下面两种方法之一进行查看。

① 采用 OPNET 与 VC 联合调试，在 VC 中通过单步执行查看。

② 在 ODB 调试过程中用 next 指令配合 ltrace 和 fulltrace 指令逐个观察程序运行情况。

仿真一个简单场景包含了所需要仿真的所有内容，如果需要扩展系统，看似只需要把相同类型的节点复制粘贴然后进行仿真即可，但是在运行的时候可能会出现无效内存访问(Invalid Memory Access)的错误，这种错误是隐藏的，所以很难发现，需要花费很长的时间来寻找，这也是导致系统稳健性不强的潜在因素。出现这类错误的原因在于内存读写越界，仿真过程中内存是不断变化的，出错的位置也在不断变化，而且当前位置的错误也不一定是出错的原因，所以必须仔细查看每个模块在开始时内存的分配方式。碰到这个问题时，可以在保证仿真可以完成的前提下，在仿真属性中选择“Advanced”→“Profiling”→“Collect detailed profiling for function”选项，之后出现每一个内存使用的统计，如图 9-51 所示，可以从这些统计中发现哪些函数占用的内存最多，哪个函数编写不够好。也可以通过 utilities→Memory Usage 物件得到一段时间内使用了多少内存，从而看出有无不正常地增加内存使用率。

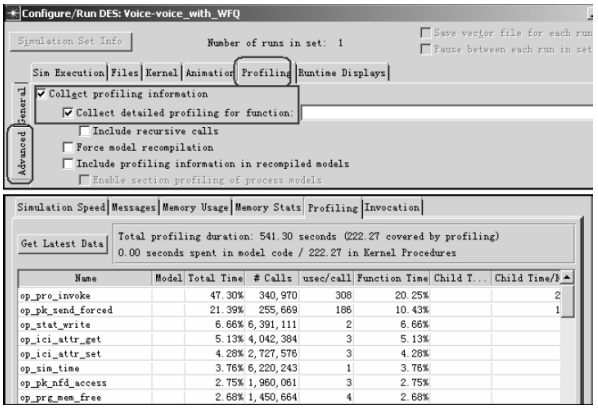


图 9-51　仿真完成后显示的函数被调用信息

（2）编译连接错误

```
<<< Recoverable Error >>>
Object repository construction failed
due to errors encountered by the binder program (bind_so_msvc)
-----
<<< Program Abort >>>
Error encountered rebuilding repository -- unable to proceed
```

出现这个错误有以下几种可能性。

1）Pipeline Stage (C code)文件名与函数名不一样，此时改为同名即可。

2）在进程模块中有一个外部函数无法定位，通过进程模型编辑器选择“File”→“Declare External Files”选项，然后选中含有该外部函数的外部文件即可。

3）外部文件用到一个无法定位的函数，这时要查看是否漏掉 include 需要用到的头文件。

（3）程序转移错误

```
<<<Program Abort>>>
No true transitions from state ()
T(), EV(), MOD(), PROC (sim_pro_err_transit)
-----
<<< Recoverable Error >>>
```

```
Unable to execute intrpt at process (0)
Process is already within an invocation.
T(), EV(), MOD(), PROC (sim_pro_err_transit)
```

程序转移错误就是状态程序执行完毕找不到出口，有限状态机要求在任意条件下每个状态执行完毕后都要发生转移。出现这种情况有以下几个原因。

1）没有转移条件及相应的状态，如图 9-52 所示，开始仅有一个 init 状态，此时需要增加一个非强制状态。

2）没有满足条件的转移，需要增加默认转移，这是为存在不满足转移条件时定义的转移。

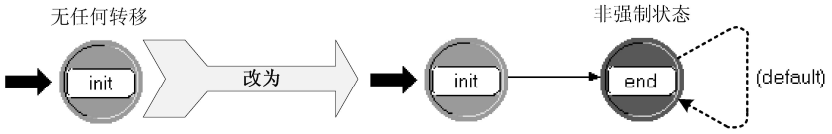


图 9-52　增加非强制状态

小结

基于 OPNET 的通信网络仿真软件通过计算机建立网络设备和网络链路的模型，利用数学建模和统计分析的方法模拟网络承载能力，从而获取网络设计及优化所需要的网络特性参数，为网络的规划和设计者提供了有效的手段，受到越来越多研究人员的青睐。

本章的目的是使读者在了解仿真技术的基础上，了解 OPNET 的发展，掌握其基本操作。本章重点为 OPNET 的运行环境、OPNET Modeler 编程基础和 OPNET 调试，希望同学们掌握这些知识。

参考文献

[1] 龙华．OPNET Modeler 与计算机网络仿真[M]. 西安：西安电子科技大学出版社，2006.
[2] 王文博，张金文．OPNET Modeler 与网络仿真[M]. 北京：人民邮电出版社，2003.
[3] 高嵩．OPNET Modeler 仿真建模大解密[M]. 北京：电子工业出版社，2010.
[4] 张铭，窦郝蕾，常春藤．OPNET Modeler 与网络仿真[M]. 北京：人民邮电出版社，2007.
[5] 周成，李丽，张裕，等．计算机通信网络与仿真[M]. 哈尔滨：哈尔滨工业大学出版社，2011.
[6] 吕跃广．通信系统仿真[M]. 北京：电子工业出版社，2010.
[7] Sethi A S，Hnatyshin V. Y. 计算机网络仿真 OPNET 实用指南[M]. 王玲芳，等译．北京：机械工业出版社，2014.
[8] 李馨，叶明．OPNET Modeler 网络建模与仿真[M]. 西安：西安电子科技大学出版社，2006.
[9] 陈敏．OPNET 网络仿真[M]. 北京：清华大学出版社，2004.
[10] 孟晨．OPNET 通信仿真开发手册[M]. 北京：国防工业出版社，2004.

第 10 章

综合案例——OPENT 编程

10.1 排队模型

OPNET Modeler 可以利用现有队列模块建立一个用来研究排队论的简单仿真模型。模型中数据源、队列及数据接收模块的节点模型都是利用处理器和队列模块构成的。

1．创建节点模型

（1）创建 FIFO 队列模型

打开节点域编辑器，利用处理器与队列模块在工作空间内构成如图 10-1 所示的结构，并按图 10-1 进行命名。

（2）模块属性编辑

在整个结构模型中，数据源为复选框，source1、source2 和 source3 节点模块，同时选中 3 个源节点模块并选中“Apply change to selected objects”复选框，再将 process model 属性设置为“simple_source”，如图 10-2 所示，则可同时改变 3 个源节点的属性。

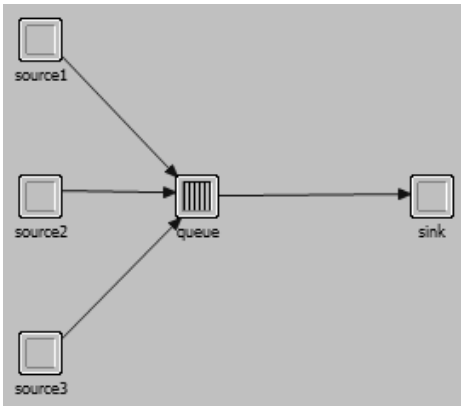


图 10-1　FIFO 队列模型

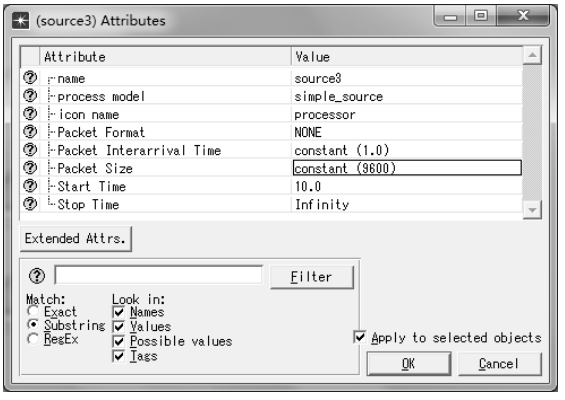


图 10-2　simple_source 的配置

在数据源的属性对话框中可以设置产生包的格式、发送包间隔的概率分布、包的大小、发送包的开始和结束时间，如图 10-2 所示。

设置节点模块 sink 的 process model 属性为 sink，表示其为数据接收模块。

把 queue 队列模块属性中 process model 设置为 acb_fifo，表示其为主动、汇集、面向比特、先进先出队列策略，如图 10-3 所示。为了容易研究子队列和容量的关系需要将 service_rate 和 subqueue 的属性设置为 promote to higher level。

（3）接口属性编辑

1）单击接口属性 Node Interfaces 的 Node types，将 mobile 和 satellite 的值更改为“no”，如图 10-4 所示。

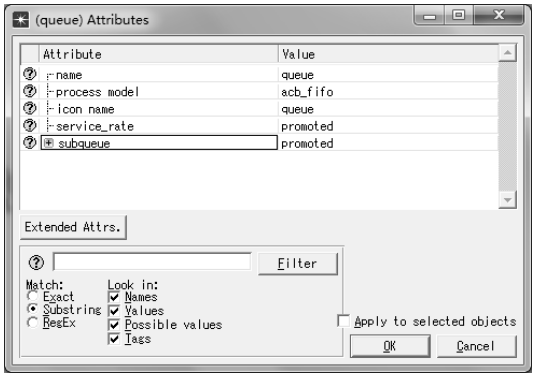


图 10-3　队列模块的属性设置

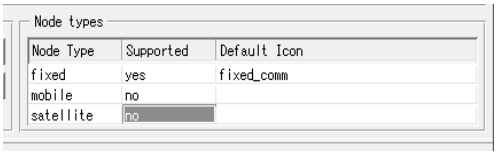


图 10-4　Node Interfaces 属性配置

2）单击接口属性 Node Statistics，弹出如图 10-5 所示的“Statistic Promotion:unnamed”对话框。单击对话框中的 Orig.Name 后出现节点模型能产生的统计量，如图 10-6 所示。

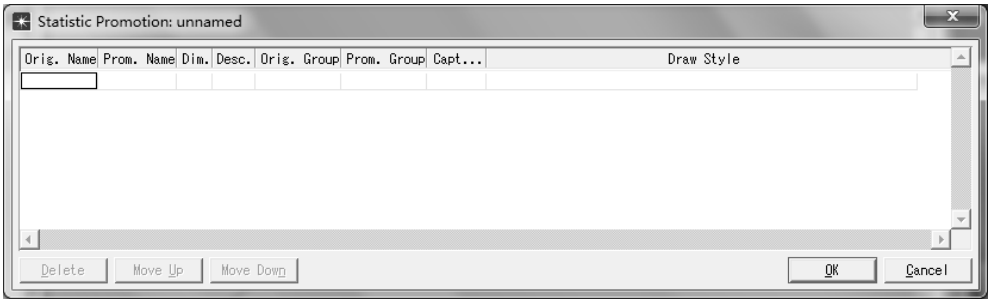


图 10-5　提升节点统计量

本实验选择的统计量包括数据源发送包情况和队列大小、时延及过流，即选择图 10-6 中的部分。单击“Promote”按钮，为了保证 Prom.group 不含有相同的统计名，在弹出的下拉列表中改变 Prom.Name 的名称，如图 10-7 所示。

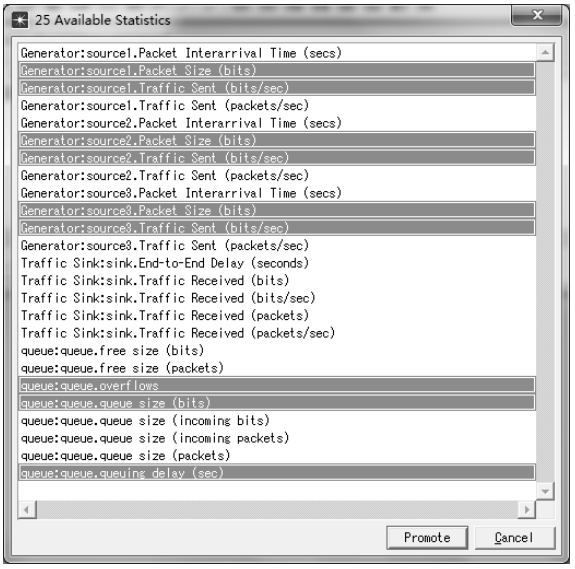


图 10-6　选择节点统计量

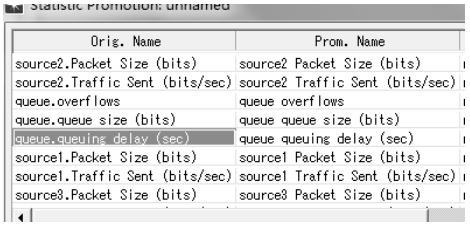


图 10-7　Orig.Name 与 Prom.Name 的关系

3）改变数据收集模式。在本次仿真中确定收集所有的统计数据，单击如图 10-5 所示的“Capture mode”按钮，弹出如图 10-8 所示的对话框，选中“Advanced”复选框，在“Capture mode”下拉列表中选择“all values”选项。

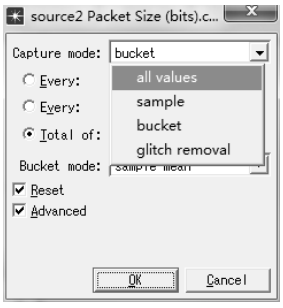


图 10-8　选择数据收集模式

（4）保存节点

保存节点并命名为 b_acb_fifo。因为仿真必须在网络域中进行，所以需要节点模型映射到网络域中。

2．映射模型

（1）建立项目环境

1）打开项目编辑器，建立新项目，选择“File”→“New”→“Project”选项。

2）设置项目名为 b_queue_disciplines，场景名为 acb-fifo，单击“OK”按钮。

3）单击“Quit”按钮。

（2）注册对象面板

1）打开对象面板，单击“Configure palette”按钮。

2）单击“Clear”按钮，将对象平台中的其他平台清除。

3）单击“Node Model”按钮，在弹出的节点模块中选择“b_acb_fifo”选项，将 not included 更改为 included，单击“OK”按钮，如图 10-9 所示。

4）在 Configure palette 平台上选择“Save as”选项，保存注册对象。此时，对象平台将出现如图 10-10 所示的建模对象。

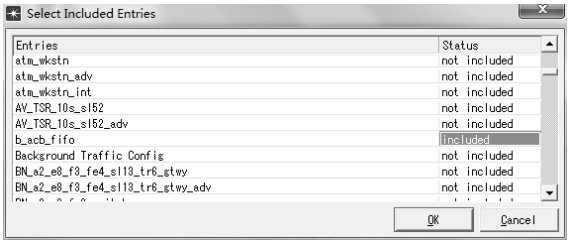


图 10-9　选择注册对象

图 10-10　新建对象平台

（3）建立项目并编辑属性

将对象 b_acb_fifo 加入到项目空间中，而出现的节点就是 FIFO 队列模型在网络域中的表现形式。

如图 10-11 所示，选中 queue 节点模块并右击，在弹出的快捷菜单中，选择“属性”选项，将 queue.service_rate 设置为 9600，将 queue.subqueue→Rows 的值设置为 1，设置 subqueue 的 pk capacity 为 infinity，即子队列具有无限存储空间，单击“Ok”按钮。

3．选择统计量

选中节点并右击，在弹出的快捷菜单中选择“Choose individual DES statistics”→“Node Statistics”选项，选择统计量，如图 10-12 所示。

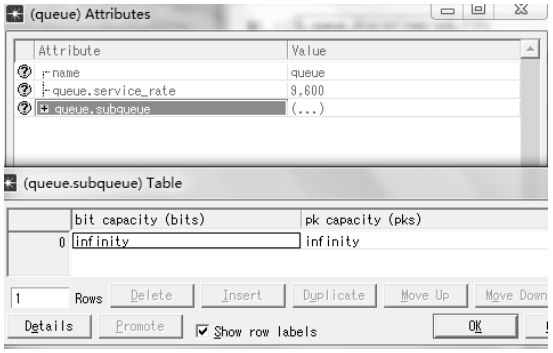


图 10-11　网络域中属性的设置

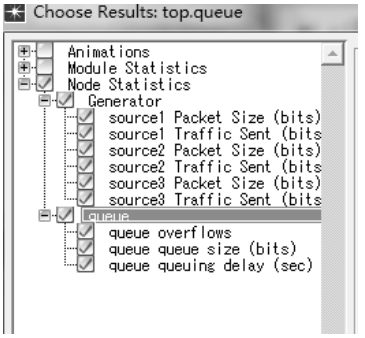


图 10-12　选择网络统计量

4．建立容量有限的队列模型

选择“Scenaio”→“Duplicate Scenaio”选项，设置场景名为 acb_fifo_finite，如图 10-13 所示，将节点 queue attribute 的属性子队列容量设置为有限值。

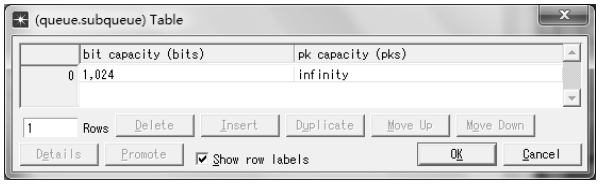


图 10-13　设置子队列容量为有限值

5．仿真及结果分析

（1）运行仿真

在两个场景下分别运行仿真，设置 Sim Duration 为 1 hour。

（2）结果观察

选择“DES”→“Result”→“View Statistics…”→“Discrete Event Graph”选项，选中相应的统计并进行观察。

这里只选择 overflows 进行比较，如图 10-14 所示。

通过仿真结果可以得到在队列容量无限和队列有限两种情况下的过流，前者几乎没有过流数据，后者则出现较大的过流。

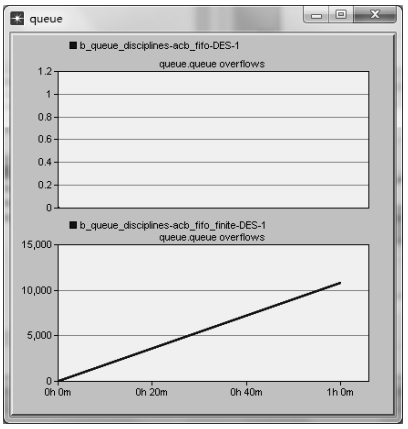


图 10-14　队列容量有限和无限时的 overflows 的统计

10.2　无线建模

本实验要完成的目标如下。

1）建立具有方向图的的天线模型和全向天线模型。

2）利用数据节点模块、无线发射和天线节点模块建立一个发送节点，天线节点模块在仿真时分别使其工作在集中定向发送天线和全向天线的模式下。

3）利用接收天线模块、无线接收机、接收机处理模块和定向天线协同工作的附加处理器模块建立一个接收节点。

4）建立一个移动干扰节点以产生无线噪声，并通过设置其轨道参数来干扰接收机。

1．建立天线

建立天线时，phi 的分片间隔采用默认的 5°，此时共有 36 个分片。

1）选择“File”→“New”→“Antenna Patten”选项，并单击“OK”按钮，弹出天线编辑器。

2）右击工作空间，设置“Set Phi Plane”中 Pin 的值为 5.00；先选择 Set Ordinate Upper Bound，再选择定义天线增益上限为 201dB；Set Ordinate Lower Bound 定义天线增益下限为 199dB。

3）在 200dB、θ=0 处单击，然后在θ=355°处再次单击，即可定义天线增益。

4）当 phi=0.00 时，进行同样的操作。单击“$1/\sum$”按钮进行归一化，可得到归一化天线增益图，如图 10-15 所示。保存并将其命名为 b_mrt_cone。

5）全向天线在各个方向上的发射增益完全一样，所以 pin 的值需要从 0.00 变化到 175.00，并且增益上线为 0.50，下限为 - 0.50。图 10-16 所示为全向天线归一化增益图。

图 10-15　b_mtr_cone 归一化天线增益

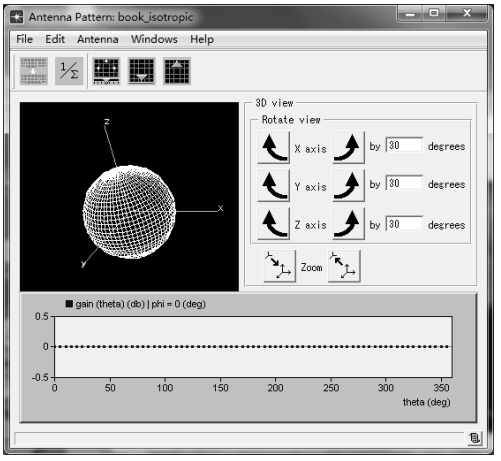


图 10-16　全向天线归一化增益图

6）保存并将其命名为 b_isotropic，关闭天线模型编辑器。

2．创建天线指向处理器

先计算出发送机的位置，然后设置天线模块的目标属性，由于天线模块不接收中断，所以可以设置为非强制状态。

1）选择“File”→“New”→“Process Model”选项；单击“OK”按钮，弹出进程编辑器。

2）在工作空间中建立命名为 point 的进程模块，模块为非强制状态。

3）建立一个指向状态自身的转移线，编辑状态转移线的属性，将 condition 设定为 default。

4）双击 point 进程模块的进入代码，在弹出的进入代码编辑器中选择“File”→“Import”选项，如图 10-17 所示。

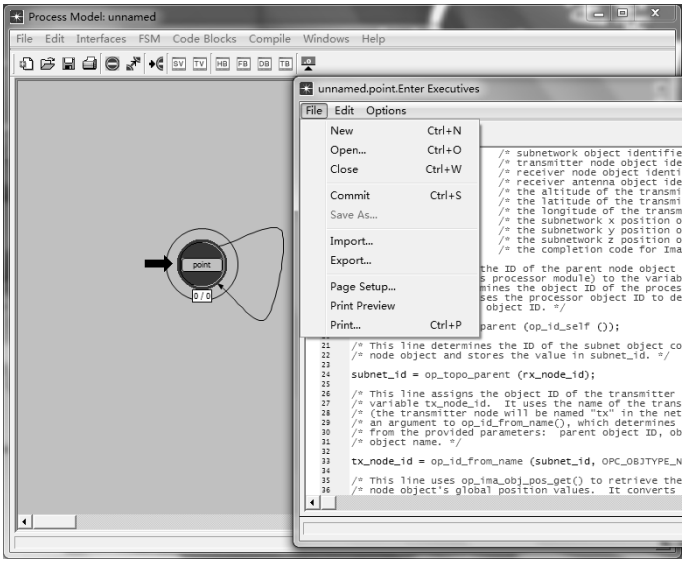


图 10-17　代码编辑器

图 10-18　引入程序

5）在安装目录中找出<reldir>/models/std/tutorial_rep/modeler/mrt_ex 文件，单击“打开”按钮后完成输入程序到进程模块的操作，如图 10-18 所示。在代码编辑器中选择“File”→“Save”选项，保存文件。

6）选择“Interfaces”→“Process Interfaces”选项，弹出进程接口对话框，将 begsim intrpt 设置为 enabled，并将所有状态的属性都设定为 hidden，保存设置。

7）编译进程模型，并将其命名为 book_mrt_rx_point。

3．建立发送节点的节点模型

包发生器模块、无线发送机模块和天线模块构成了发送节点。假设包发生器每秒产生一个 1024bit 的数据包，数据包通过包流线发送到发送机上，在信道上的传输速率为 1024b/s，最后包被传递到天线模块中。

根据假设，完成发送节点建模的操作步骤如下。

1）选择“File”→“New”→“Node Model”选项。

2）建立如图 10-19 所示的发送节点的节点模型，并按如图 10-19 命名。

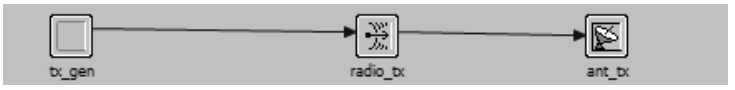


图 10-19　发送节点的节点模型

3）设置属性。将 tx_gen 的 process model 设置为“simple_source”，其他属性配置如图 10-20 所示。

将 radio__tx 节点模块的 channel 属性中的 power 提升，其他管道阶段配置选择默认值，如图 10-21 所示。

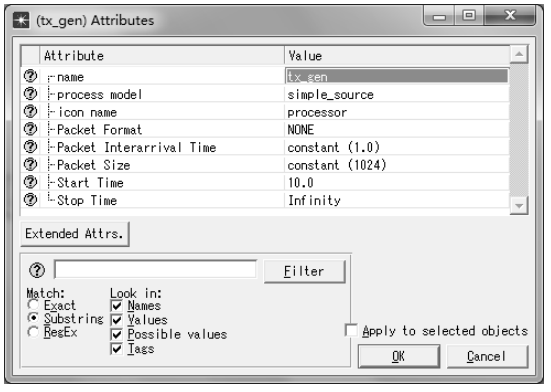


图 10-20　数据发送模块的属性配置

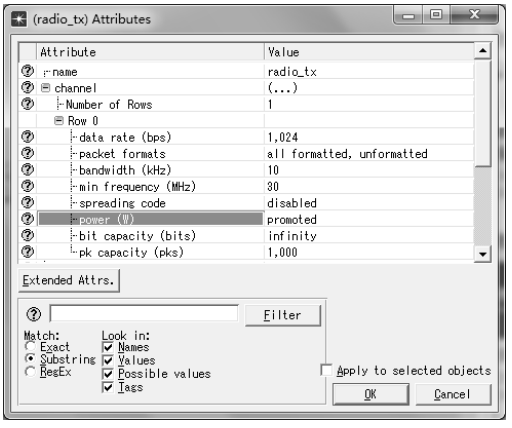


图 10-21　节点模块的属性配置

将 ant_tx 的 pattern 属性设置为 book_mrt_cone，即选择天线设置为本实验最初建立的天线模型，如图 10-22 所示。

选择“Interfaces”→“Node Interfaces”选项，在节点类型中将 satellite type 和 mobile 的 support

值设置为“no”，在属性列表中，将 altitude 设置为 0.003。除了 radio_tx channel1[0].power 属性外，其他属性状态设定为“hidden”。在关键字一栏中，输入 mrt，如图 10-23 所示。

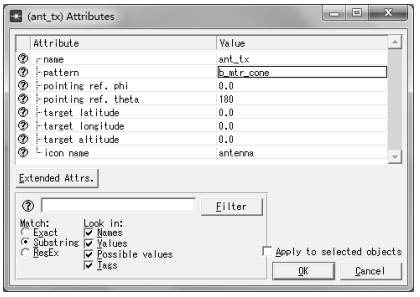


图 10-22　天线模型的属性

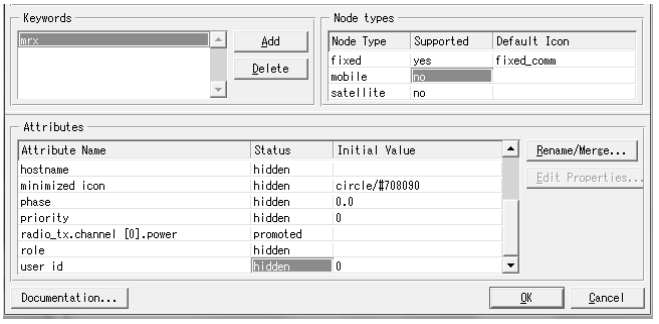


图 10-23　发送节点模型的接口属性

4）保存模型，并将其命名为 b_mrt_tx。

4．干扰节点

干扰节点的作用就是向网络中引入噪声，干扰节点也包含包发送模块、无线发送机和天线模块，这与静止的发送节点相似，但是不同之处在于信道功率与调制方式，因此可从静止发送节点进行修改得到干扰节点。

1）打开已经建立的 book_mrt_tx 节点模型。

2）将其另存为 book_mrt_jam。

3）右击 radio_tx，在弹出的快捷菜单中选择“属性”选项，将 modulation 属性设置为“jammod”，如图 10-24 所示。

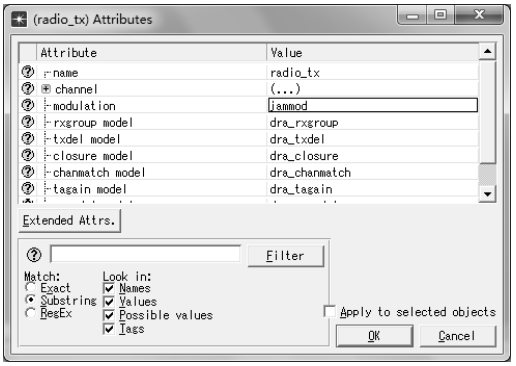


图 10-24　干扰节点模型的属性

4）选择“Node Interfaces”→“Interfaces”选项，修改 Node Types 使其支持 mobile，保存节点模型。

5．接收节点

接收天线、无线接收机、包接收模块和一个指向处理器模块构成了接收节点，指向处理器

模块用于控制天线方向。

1）选择“Edit”→“Clear Model”选项。

2）建立如图 10-25 所示的节点模型，并进行节点模型命名。

3）编辑其属性。

rx_point 是指向处理器模块，选中“rx_point”并右击，在弹出的快捷菜单中选择“Edit Attributtes”选项，将 process model 设置为“b_mrt_rx_point”，如图 10-26 所示。

radio_rx 是无线接收机，右击“radio_rx”，在弹出的快捷菜单中选择“属性”选项，弹出属性对话框。将 error model 设置为“dra_error_all_stats”。ecc threshold 表示接收机的错误门限，0.0 表示当没有错误比特时，包才会被接收。其余管道阶段的设置取默认值，如图 10-27 所示。

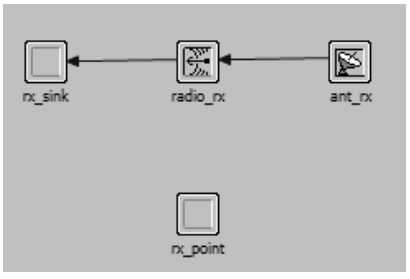


图 10-25　无线信号接收节点

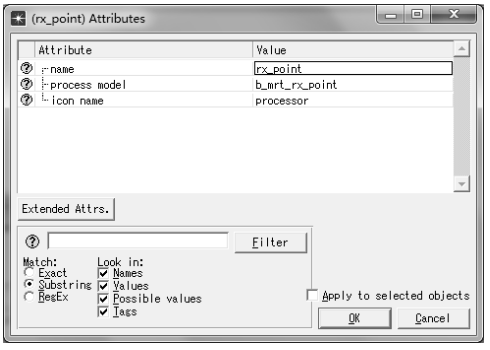


图 10-26　指向处理器模块的属性

ant_rx 是接收天线模块，右击“ant_rx”，在弹出的快捷菜单中选择“属性”选项，弹出属性对话框；将“pattern”的属性改为“promoted”，如图 10-28 所示。

4）节点模块 rx_sink 负责信号处理，此处将 process model 设置为“sink”，模块完成包的销毁。

5）建立节点接口属性。选择“Interfaces”→“Node Interfaces”选项，在 Node Types 中，令其不支持 satellite type 和 mobile；在属性列表中，将 altitude 设置为 0.003。除了将属性 ant_rx.pattern 设置为 promoted 之外，其他属性的 Status 值均设置为 hidden。

6）单击“OK”按钮，保存更改。

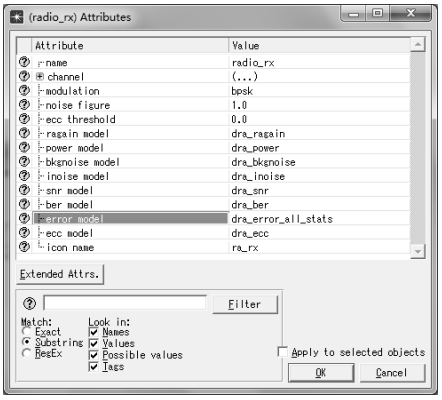


图 10-27　无线接收机的属性

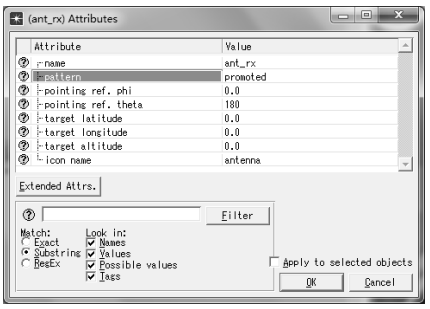


图 10-28　接收天线模块的属性

6．建立网络模型

1）选择“File”→“New”选项，选择建立新的项目。

2）将新项目命名为 b_mrt_net，场景名称为 antenna_test。

3）根据第 9 章实例的建模步骤，建立 Network Scale 为 Enterprise，Specify Size 为 10km×10km 的网络，在 View→Set Background Properties 中设置网络背景的网格大小为 1km。

4）注册对象面板。在对象面板中，选择“Configure Palette”→“Clear”→“Node Palette”选项，加入 b_mrt_jam、b_mrt_rx、b_mrt_tx，并将对象面板保存为 b_mrt_net_antenna_test，如图 10-29 所示。

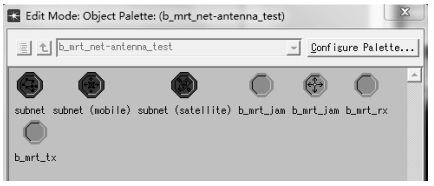


图 10-29　建立建模对象平台

5）将对象面板中的 3 个对象分别拖入工作空间，建立如图 10-30 所示的网络模型。其中每个节点的位置如下。

tx（3，4），rx（4，4），jam（0.5，3.5）。

6）在所定义的 3 个节点中，干扰节点将会运动地移进发送和接收节点。这时需要定义其轨迹，即选择“Topology”→“Fine Trajectory”选项，在弹出的对话框中，定义 Trajectory type 为“Fixed interval”，Altitude 为 0.003kilometer（s），Time step 为 0h7m0s 的轨迹，如图 10-31 所示。

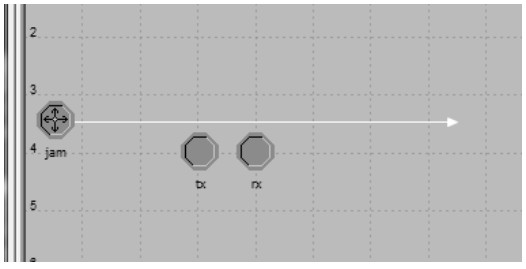


图 10-30　网络模型

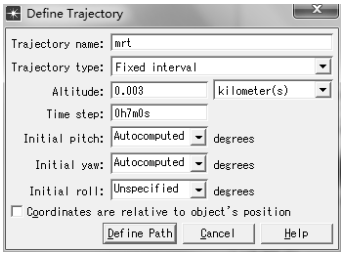


图 10-31　定义轨迹

7）单击“Define Path”按钮，用出现的带箭头的鼠标指针从 jam 节点位置拖动到（7.5，3.5）的位置。右击结束拖动，保存并将其命名为 mrt。

8）为了使轨迹更准确，右击轨迹，在弹出的快捷菜单中选择“Edit trajectory”选项， 图 10-32 所示的数据精确地定义了轨迹。

9）单击“OK”按钮，关闭对话框，并保存对项目的设置。

7．收集统计数据及仿真的运行

1）选中如图 10-33 所示的 rx 节点并右击，在弹出的快捷菜单中选择“Choose Individual Statistics”选项。选择统计数据为 Module Statistics→radio_rx.channel1[0]→bit error rate 和 Module Statistics→radio_rx.channel1[0]→radio receiver→throughput（bits/sec），如图 10-33 所示。

2）改变统计模式。右击“bit error rate”，在弹出的快捷菜单中选择“Change Collection Mode”选项，选中“Advanced”复选框，将 Capture mode 更改为“glitch removal”，如图 10-34 所示。单击“OK”按钮保存选择。

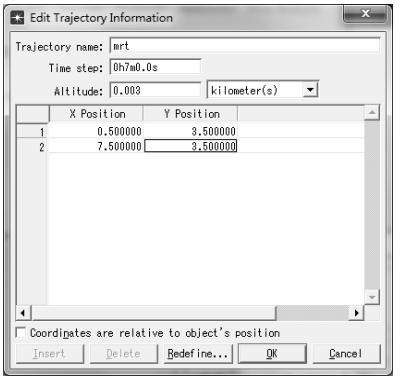


图 10-32　编辑轨迹

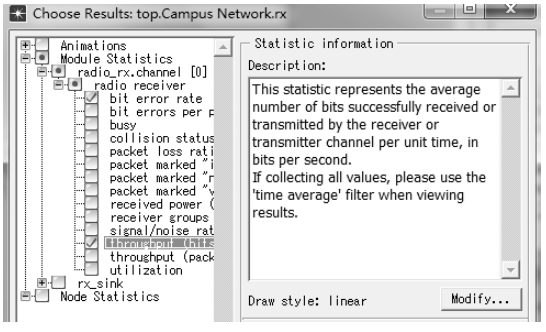


图 10-33　选择统计量

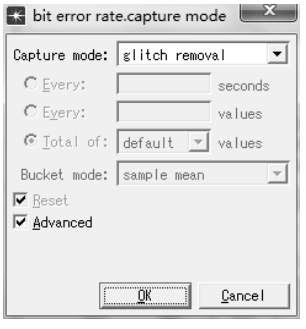


图 10-34　改变 bit error 统计收集方式

同样，右击“throughput（packet/sec）”，在弹出的快捷菜单中选择“Change Collection Mode”选项，选中“Advanced”复选框，将 Capture Mode 设置为“bucket”，并将 Bucket mode 设置为“max value”。将抽样频率设置为 10 seconds，如图 10-35 所示。注意，不要选中“Reset”复选框。

3）单击“Ok”按钮，保存所做的设置。

8．运行仿真

1）提升属性的赋值。返回项目编辑器，选择“Simulation”→“Configure Discrete Event Simulation（Advanced）”选项，打开仿真顺序编辑器。在编辑器左侧找到场景（scenario）并右击，在弹出的快捷菜单中选择“Edit Attributed”选项。

在“General”→“Object Attributes”中，单击“Add”按钮，将前面所选的 3 个提升的属性添加到仿真环境的对象属性中，如图 10-36 所示。

单击“OK”按钮后出现的 3 项属性都是在节点编辑器中提升过的属性名词，需要对其进行设置。

选择属性 ant_rx.pattern，单击“Value”按钮，在弹出的下拉列表中选择“b_isotropic”选项，单击“Set Mutiple Values”按钮，添加 b_mrt_cone，再单击“OK”按钮即可。

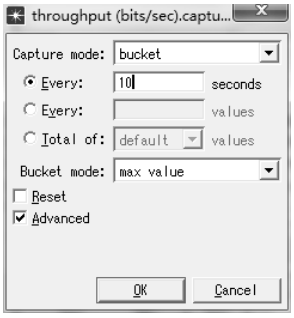


图 10-35　改变 throughput 统计收集方式

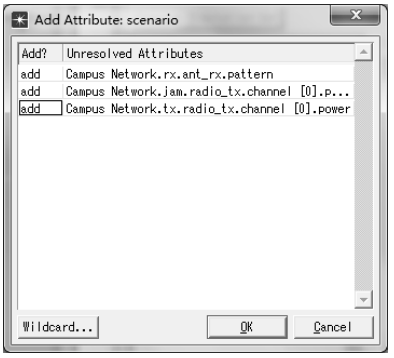


图 10-36　仿真环境中对象属性的编辑

将 jam_radio_tx.channel[0].power 设置为 20，将 tx.radio_txd1annelm_power 设置为 1。

在图 10-37 所示的对话框中显示“Number of runs：2”，这是因为在 ant_rx.pattem 属性中设置了两个取值，所以会运行两次仿真。

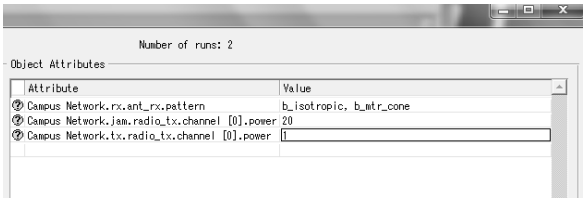


图 10-37　添加对象属性的对应值

2）设置仿真参数。在“Common”选项组中将 Seed 设置为“50”，Duration 设置为“450”seconds，Update interval 设置为“100”，如图 10-38 所示。

可以对“Common”选项组中的属性进行如下设定。

Seed　　50

Duration　　450 seconds

Update interval　　100

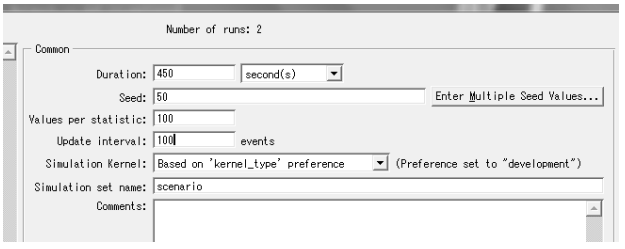


图 10-38　设置仿真参数

3）清空 repositories 属性。选择“Edit”→“Preferences”选项，将 repositories 一项清空。

4）运行仿真。

9．仿真结果的观察

1）选择“File”→“New”→“Analysis Configuration”选项。

2）观察吞吐量的影响。首先停止对误码率的选择，然后同时选择 b_mrt_net_antenna_test_1 和 b_mrt_net_antenna_test_2 中的 througput，选择 Statistics Overlaid，把 As Is 改为 average，如图 10-39 所示。

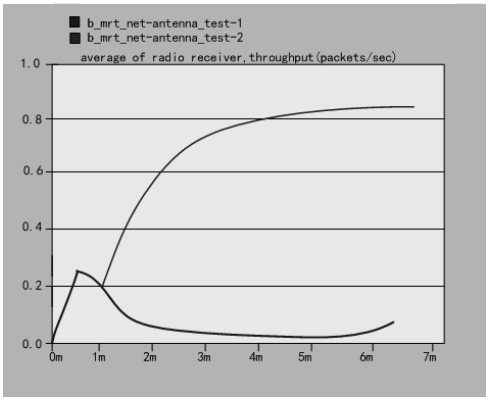


图 10-39　两种天线模式下的吞吐量

对于全向天线，接收到的平均误码率会随时间不断减小，当干扰节点和接收节点的距离逐渐增加后，误码率略有增加。而有向天线在很短的时间内干扰节点不再对接收节点产生影响，吞吐量逐渐增加。

参考文献

[1] 龙华．OPNET Modeler 与计算机网络仿真[M]. 西安：西安电子科技大学出版社，2006.
[2] 王文博，张金文．OPNET Modeler 与网络仿真[M]. 北京：人民邮电出版社，2003.
[3] 高嵩．OPNET Modeler 仿真建模大解密[M]. 北京：电子工业出版社，2010.
[4] 张铭，窦郝蕾，常春藤．OPNET Modeler 与网络仿真[M]. 北京：人民邮电出版社，2007.
[5] 周成，李丽，张裕，等．计算机通信网络与仿真[M]. 哈尔滨：哈尔滨工业大学出版社，2011.
[6] 吕跃广．通信系统仿真[M]. 北京：电子工业出版社，2010.
[7] Sethi，A．S，HnatyshinV．Y. 计算机网络仿真 OPNET 实用指南[M]. 王玲芳，等译．北京：机械工业出版社，2014.
[8] 李馨，叶明．OPNET Modeler 网络建模与仿真[M]. 西安：西安电子科技大学出版社，2006.
[9] 陈敏．OPNET 网络仿真[M]. 北京：清华大学出版社，2004.
[10] 孟晨．OPNET 通信仿真开发手册[M]. 北京：国防工业出版社，2004.

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888
传　　真：（010）88254397
E-mail：　dbqq@phei.com.cn
通信地址：北京市万寿路 173 信箱
　　电子工业出版社总编办公室
邮　　编：100036